



Quick answers to common problems

CryENGINE 3 Cookbook

Over 90 recipes written by Crytek developers for creating
third-generation real-time games



Dan Tracy

Sean Tracy

[PACKT]
PUBLISHING

CryENGINE 3 Cookbook

Over 90 recipes written by Crytek developers for creating
third-generation real-time games

Dan Tracy

Sean Tracy



BIRMINGHAM - MUMBAI

CryENGINE 3 Cookbook

Copyright © 2011 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: June 2011

Production Reference: 1170611

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-849691-06-2

www.packtpub.com

Cover Image by Sean Tracy (seanp@crytek.com)

Credits

Authors

Sean Tracy

Dan Tracy

Acquisition Editor

Steven Wilding

Development Editor

Alina Lewis

Technical Editor

Aditi Suvarna

Copy Editor

Laxmi Subramanian

Indexer

Rekha Nair

Project Coordinator

Zainab Bagasrawala

Proofreader

Aaron Nash

Graphics

Geetanjali Sawant

Production Coordinator

Shantanu Zagade

Cover Work

Shantanu Zagade

About the Authors

Sean Tracy is Crytek's Senior Field Application Engineer for the award-winning CryENGINE. He is responsible for adapting the engine and its features to individual licensees, as well as developing full technical and vertical slice demos for prospective and existing clients. Describing himself as a generalizing specialist, he also gives support directly to CryENGINE licensees, while designing and maintaining their workflows, pipelines, and development techniques.

Sean was recruited by Crytek in 2008 after working as an electronics technician for the Canadian Military. He was recruited due to his role in founding and leading the development on the award-winning total conversion project MechWarrior: Living Legends. Since then, he has been featured in numerous gaming magazines and has been invited to speak at many game-related trade shows and seminars. He is an avid gamer with extensive modding experience on titles including Never Winter Nights, the Battlefield engine Frostbite, Doom, and Quake.

I would like to thank my wife for her understanding and support throughout the process of writing this book and for her ongoing support in allowing me to do what I truly enjoy for a living. I would also like to thank my brother for co-authoring the book with me as it's a pleasure to be able to work with someone with the same love for the technology as I have. Finally, I'd like to thank Crytek and Packt for their support in allowing me to write this book and for making one of the best game engines in the market.

Dan Tracy is Crytek's Technical Level Designer for the award-winning CryENGINE and Crysis 2. He is responsible for the creation and maintenance of numerous technical features and external applications used for telemetry and optimization. Viewed as more than a level designer, Dan prides himself on pushing the envelope when it comes to improving both technical and game related designs across multiple production disciplines.

Dan was recruited by Crytek in 2009 after previously working as a Quality Assurance Technician for BioWare. He was recruited due to his pivotal role in co-founding and leading development on the award winning total conversion project MechWarrior: Living Legends. Since then, he has been featured in numerous gaming magazines and has been interviewed by multiple media outlets. Dan is a passionate gamer, but an even more passionate modder and game designer, with vast knowledge and experience with multiple engines and titles including Never Winter Nights' Aurora, Battlefield's engine Frostbite, Unreal 3, and CryENGINE. This is Dan's first book.

I would like to thank my friends and family for giving me their support during the crunch time of the Crysis 2 production, which also paralleled the creation of this book. If it wasn't for them, this wouldn't have been possible. I would also like to thank my brother for co-authoring the book with me as well as Crytek for providing me with this amazing opportunity to share my knowledge of CryENGINE with the world. Finally, I'd like to thank Packt for their support and setting this whole project in motion and publishing my first ever book.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and, as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read, and search across Packt's entire library of books.

Why subscribe?

- ▶ Fully searchable across every book published by Packt
- ▶ Copy and paste, print, and bookmark content
- ▶ On demand and accessible via web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	1
Chapter 1: CryENGINE 3: Getting Started	5
Introduction	5
Opening a level in the CryENGINE 3 Sandbox	6
Navigating a level with the Sandbox Camera	8
Setting up a personalized toolset layout	12
How to customize toolbars and menus	17
Using the Rollup Bar	21
Selecting and browsing level objects	25
Restoring the CryENGINE 3 default settings	28
Chapter 2: Sandbox Basics	31
Introduction	31
Creating a new level	32
Generating a procedural terrain	34
Terrain sculpting	37
Setting up the terrain texture	41
Placing the objects in the world	43
Refining the object placement	44
Utilizing the layers for multiple developer collaboration	47
Switching to game mode	49
Saving your level	50
Exporting to an engine	52
Essential game objects	53
Running a map from the Launcher	54
Chapter 3: Basic Level Layout	55
Introduction	55
Making basic shapes with the Solids tool	56
Editing and merging solids	57

Grouping the objects	60
Utilizing the Geom entities instead of brushes	61
Road construction	62
Painting vegetation	64
Breaking up tiling with Decals	67
Making caves with Voxels	69
Creating Prefabs to store in external libraries	72
Chapter 4: Environment Creation	75
Introduction	75
Creating your first time of day using the basic parameters	76
Adjusting the terrain lighting	83
Using the real-time Global Illumination	86
Editing HDR lighting and the effects for flares	89
Creating a global volumetric fog	92
Creating a night scene with time of day parameters	95
Color grading your level	97
Creating a photo realistic ocean	99
Improving your sky with clouds	102
Making it rain in your level	105
Chapter 5: Basic Artificial Intelligence	109
Introduction	109
Placing the enemy AI	110
Generating the AI navigation	111
Forbidden boundaries	112
Forbidden areas	113
Setting up the interior navigation	114
Debugging the AI triangulation	119
Narrowing the AI's FOV to allow attacks from behind	121
Respawning AI	122
Chapter 6: Asset Creation	127
Introduction	127
Installing the CryENGINE 3 plugin for 3D Studio Max	128
Creating textures using CryTIF	131
Setting up units to match CryENGINE in 3ds	135
Basic material setup in 3ds	138
Creating and exporting static objects	143
Creating and exporting destroyable objects	149
Using advanced material editor parameters to create animation	153
Creating new material effects	155
Creating image-based lighting	159

Chapter 7: Characters and Animation	163
Introduction	163
Creating skinned characters for the CryENGINE	164
Ragdoll and physics for characters	173
Creating animation for your character	178
Previewing animations and characters for Sandbox	185
Creating upper body only animations	188
Creating locomotion animations	191
Animating rigid body geometry data	195
Chapter 8: Creating Vehicles	199
Introduction	199
Creating a new car mesh (CGA)	199
Creating a new car XML	205
Giving more speed to the car	211
Increasing the mass to push objects with the car	212
Defining a sitting location	213
Setting up multiple cameras for the car	215
Need for a machine gun	218
Giving the car a weak spot	219
Chapter 9: Game Logic	223
Introduction	223
How to beam the player to a tag point from a trigger	223
Making the AI go to a location when the player enters a proximity trigger	227
Debugging the Flow Graph	229
Creating a kill counter	230
Rewarding the player for reaching a kill goal	232
Displaying the player's health through a Flow Graph	234
Changing the player camera through key input	236
Creating a countdown timer	238
Chapter 10: Track View and Cut-Scenes	241
Introduction	241
Creating a new Track View sequence	242
Animating a camera in the Track View	246
Triggering a sequence using the Flow Graph	251
Animating entities in the Track View	254
Playing animations on entities in the Track View	258
Using console variables (CVars) in the Track View	260
Using track events	263

Chapter 11: Fun Physics	267
Introduction	267
Low gravity	267
Hangman on a rope	269
Tornadoes	272
Constraints	273
Wrecking ball	275
Rock slide	277
Chapter 12: Profiling and Improving Performance	281
Introduction	281
Profiling performance in the Sandbox	282
Saving level statics	285
Enabling the debug draw modes	290
Optimizing the levels with VisAreas and portals	294
Using light boxes and light areas	297
Activating and deactivating the layers	298
Index	301

Preface

With the overall complexity involved in creating games becoming exceedingly difficult and expensive with every successive console generation, many game developers have turned to middleware engines, such as the CryENGINE, that offer a complete pipeline for the game development process. CryENGINE is a perfect fit for most developers as it allows users to create their content quickly and easily and thus, allow games to meet and exceed current generation quality standards and still be created by fewer and fewer people.

As CryENGINE 3 is globally recognized as one of the world's most powerful real-time middleware development platforms, with this book we will deliver the best of what the engine has to offer. Through the use of CryENGINE's intuitive and powerful toolset, named Sandbox, designers, artists, animators, and even programmers will be treated to real-time creation and iteration tools for bringing their visions to life.

What this book covers

Chapter 1, Getting Started, helps you set up the entire CryENGINE 3 Software Development Kit, which can be a difficult task. This chapter will guide you through the stages in setting up the required folder structure and how to set up your layout for the Sandbox Editor.

Chapter 2, Sandbox Basics, helps you to learn the basic and most commonly used features provided by Sandbox.

Chapter 3, Basic Level Layout, helps you create your first Level Layout within the Sandbox Editor and learn some of the more advanced techniques used by designers for object placement and manipulation.

Chapter 4, Environment Creation, utilizes the CryENGINE 3 rendering tools to create photorealistic environments.

Chapter 5, Basic Artificial Intelligence, helps you learn the basics of how to use AI to navigate in your levels.

Chapter 6, Asset Creation, helps you learn the pipeline of asset creation and export your 3D models to the CryENGINE format.

Chapter 7, Characters and Animation, describes how to create new characters to be used in the engine along with your own custom animations.

Chapter 8, Creating Vehicles, describes how to create a new vehicle from scratch and set up the entity code required so your players can drive.

Chapter 9, Game Logic, helps you to get started with the highly versatile Flow Graph Editor and create many useful scripts used in the level.

Chapter 10, Track View and Cut-Scenes, helps you to learn how to create your own cinematic cut-scenes within CryENGINE.

Chapter 11, Fun Physics, describes how to set up some enjoyable physics contraptions using CryENGINE 3's physics system.

Chapter 12, Profiling and Improving Performance, helps you to learn the tools behind profiling your levels and discover the best methods for improving performance.

What you need for this book

The Software Development Kit version of the CryENGINE is used for all examples in this book, thus, the reader should have a version of the development kit to be able to follow the recipes contained in this book.

Who this book is for

This book is written with the casual and professional developer in mind. With that said, it is important that the readers have some fundamental knowledge of some Digital Content Creation Tools, such as Photoshop and 3d Studio Max. Though not a fundamental requirement, having some basic knowledge of real-time graphics software and, consequently, the terminology used will make the goal of these recipes more clear.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "The level must also be inside of your `Build` folder."

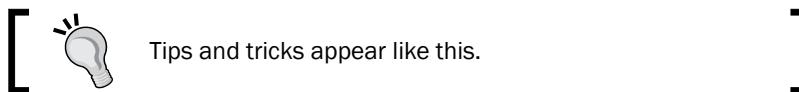
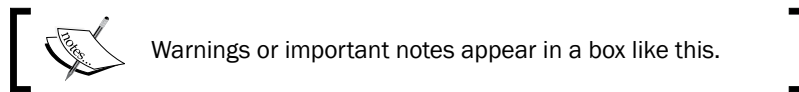
A block of code is set as follows:

```
<DamageMultipliers>
  <DamageMultiplier damageType="bullet" multiplier="0.125"/>
  <DamageMultiplier damageType="collision" multiplier="1"/>
</DamageMultipliers>
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
<DamageMultipliers>
  <DamageMultiplier damageType="bullet" multiplier="0.125"/>
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "To open an existing level, we need to access the **File** menu."



Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or e-mail suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code and colored graphics for this book

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

CryENGINE 3: Getting Started

In this chapter, we will cover:

- ▶ Opening a level using the CryENGINE 3 Sandbox
- ▶ Navigating a level with the Sandbox Camera
- ▶ Setting up a personalized toolset layout
- ▶ How to customize toolbars and menus
- ▶ Using the Rollup Bar
- ▶ Selecting and browsing level objects
- ▶ Restoring the CryENGINE 3 default settings

Introduction

The main focus of this particular chapter will be in getting the **CryENGINE 3 Software Development Kit** installed and having you up and editing a level in the Sandbox editor right away! One of the key things to keep in mind when learning a game compositing tool such as Sandbox is to remember to experiment and have fun! It is important not to forget that most of us (game developers) are trying to make things fun and not dreary and dull.

With such a powerful toolset waiting for you to dive in, let's get right to it!

Opening a level in the CryENGINE 3 Sandbox

As most people involved in the game's development process should be familiar with opening levels, this section will take you through the relatively straightforward task of opening a level within the CryENGINE 3 Sandbox editing tool.

Getting ready

Having already located the `Editor.exe` in either your `bin32` or `bin64` folders, it will now be started in this section.

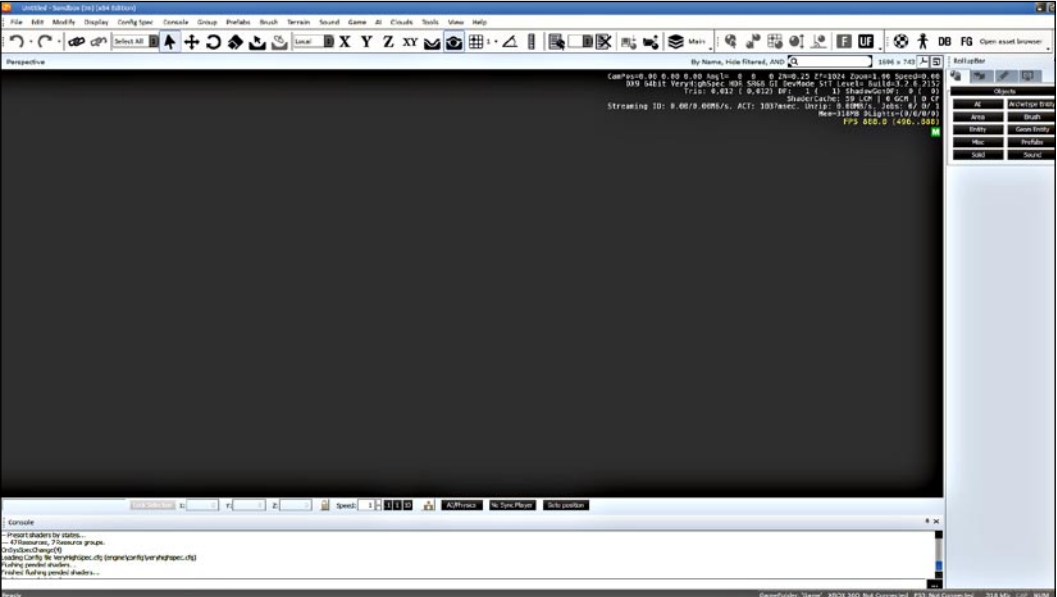


If a level is not already loaded, the editor's subsystems can still access assets and resources from your game. Keep this in mind as some tasks don't require the loading of a level.

How to do it...

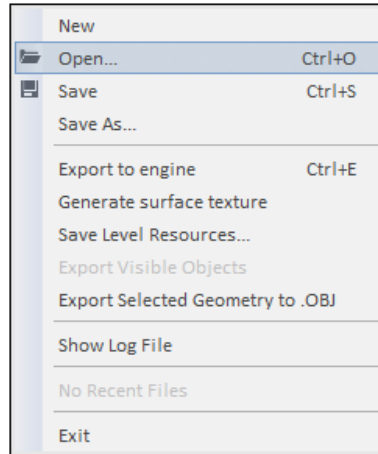
Let's get to opening a level:

1. Launch the `Editor.exe` located in either the `Bin32` or `Bin64` folder. You will be presented with an interface, as shown in the following screenshot:



2. As there are already example levels installed with the SDK, we can open them.

- To open an existing level, we need to access the **File** menu.



- The **File** menu includes commands related to the handling of level files such as opening, saving, showing log files, and a list of recently-loaded levels.
- As we want to open a pre-existing level, choose the **Open** option.
- You will then be presented with a browser window defaulting to the `CryENGINE3/game/levels` folder.
- Browse to `Forest` and open the folder.
- Within the folder there is a `Forest.cry` file that contains raw level data for editing.
- Open the `Forest.cry` file.

Name	Date modified	Type	Size
console	26.10.2010 17:01	File folder	
Layers	25.10.2010 09:24	File folder	
Forest.cry	10.09.2010 00:10	CRY File	10.323 KB

The editor will now start to load this level for you to start exploring!

How it works...

The editor reads the `.cry` files and can also access the subfolder `layers` within the level folder.

As the level loads, it reads the information present in the `.cry` file.

There's more...

You may want to know what the `.cry` file is composed of or even how to apply console command changes to individual levels.

What is a `.cry` file?

A `.cry` file is the principle level editing format for all levels built in the CryENGINE. It is actually an archive comprised of binary and XML data that is used only by the editor. You can open `.cry` files in the editor, or you can open them with an appropriate archiving program such as WinRAR.

Using a `level.cfg`

Similar to the `system.cfg`, the `level.cfg` is a file that is executed upon the loading of a level. The `level.cfg` can simply be stored in the level's directory. You may add console variables or level-specific configurations to this file.

See also

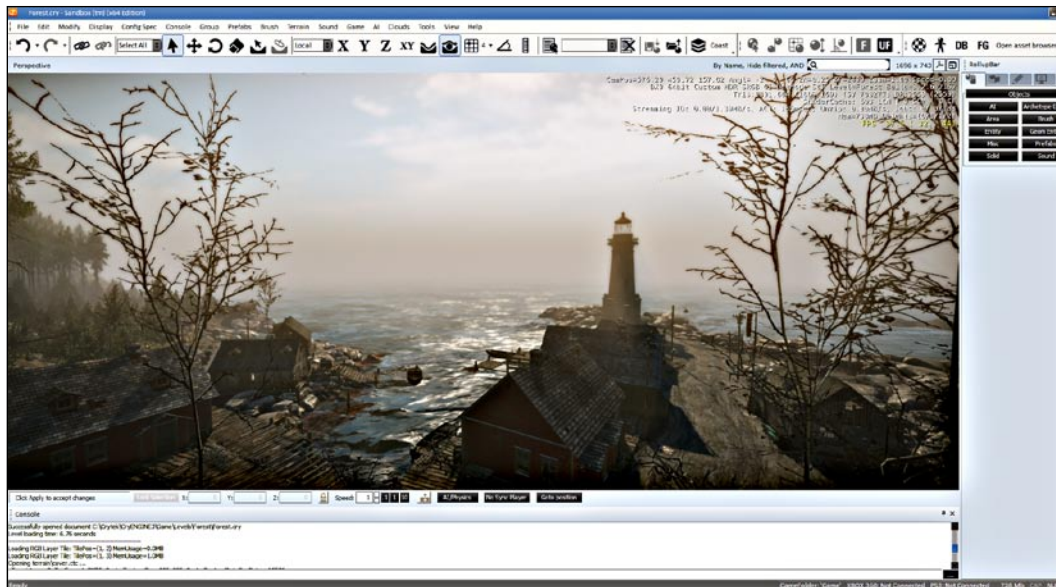
- ▶ Having launched the Sandbox, you can continue to the *Navigating a level with the Sandbox Camera* recipe in this chapter
- ▶ To get right to modifying a level, go to the *Selecting and Browsing level objects* recipe later in this chapter

Navigating a level with the Sandbox Camera

The ability to intuitively navigate levels is a basic skill that all developers should be familiar with. Thankfully, this interface is quite intuitive to anyone who is already familiar with the WASD control scheme popular in most *First Person Shooters Games* developed on the PC.

Getting ready

You should have already opened a level from the **CryENGINE 3 Software Development Kit** content and seen a perspective viewport displaying the level.



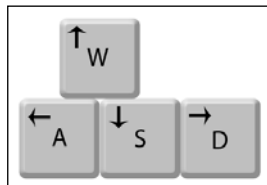
The window where you can see the level is called the **Perspective Viewport window**. It is used as the main window to view and navigate your level. This is where a large majority of your level will be created and common tasks such as object placement, terrain editing, and in-editor play testing will be performed.

How to do it...

The first step to interacting with the loaded level is to practice moving in the Perspective Viewport window.

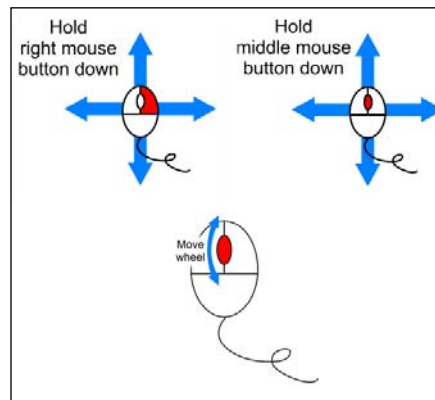


Sandbox is designed to be ergonomic for both left and right-handed users. In this example, we use the WASD control scheme, but the arrow keys are also supported for movement of the camera.



1. Press **W** to move forwards.
2. Then press **S** to move backwards.

3. **A** is pressed to move or strafe left.
4. Finally, **D** is pressed to move or strafe right.
5. Now you have learned to move the camera on its main axes, it's time to adjust the rotation of the camera.
6. When the viewport is the active window, hold down the right mouse button on your mouse and move the mouse pointer to turn the view.
7. You can also hold down the middle mouse button and move the mouse pointer to pan the view.
8. Roll the middle mouse button wheel to move the view forward or backward.
9. Finally, you can hold down *Shift* to double the speed of the viewport movements.



How it works...

The Viewport allows for a huge diversity of views and layouts for you to view your level; the perspective view is just one of many. The perspective view is commonly used as it displays the output of the render engine. It also presents you a view of your level using the standard camera perspective, showing all level geometry, lighting, and effects.

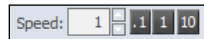
To experiment further with the viewport, note that it can also render subsystems and their toolsets such as flow graph, or character editor.

There's more...

You will likely want to adjust the movement speed and how to customize the viewport to your individual use. You can also split the viewport in multiple different views, which is discussed further.

Viewport movement speed control

The **Speed** input is used to increase or decrease the movement speed of all the movements you make in the main Perspective Viewport.



The three buttons to the right of the **Speed** inputs are quick links to the **.1**, **1**, and **10** speeds.

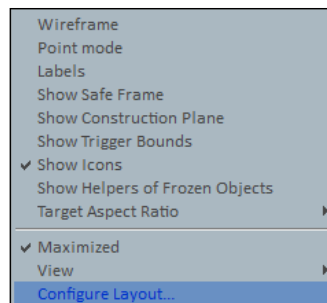
Under Views you can adjust the viewport to view different aspects of your level

Top View, Front, and Left views will show their respective aspects of your level, consisting of bounding boxes and line-based helpers. It should be noted that geometry is not drawn.

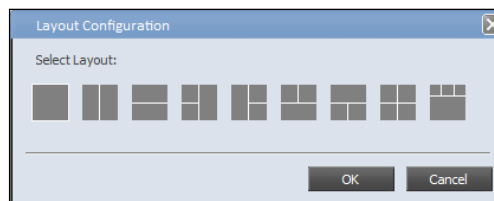
Map view shows an overhead map of your level with helper, terrain, and texture information pertaining to your level.

Splitting the main viewport to several subviewports

Individual users can customize the layout and set viewing options specific to their needs using the viewport menu accessed by right-clicking on the viewports header.



The Layout Configuration window can be opened from the viewport header under **Configure Layout**. Once selected, you will be able to select one of the preset configurations to arrange the windows of the Sandbox editor into multiple viewport configurations. It should be recognized that in multiple viewport configurations some rendering effects may be disabled or performance may be reduced.



See also

- ▶ To start building your own objects immediately, go to the *Making basic shapes with the Solids Tool* recipe in *Chapter 3, Basic Level Layout*
- ▶ To modify the terrain of the current level, go to the *Terrain Sculpting* recipe in *Chapter 2, Sandbox Basics*

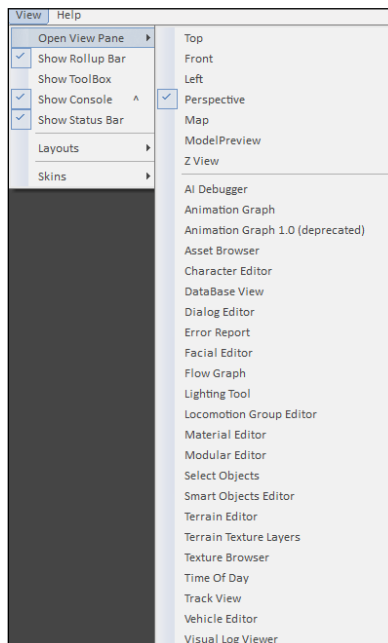
Setting up a personalized toolset layout

It would suffice to say that every user of Sandbox will have different preferences to how different views and toolsets should be distributed on screen. The CryENGINE 3 Sandbox allows for this kind of user-based customization and this recipe will take you through the use of some of the built-in tools for customizing your interface.

Getting ready

Before starting, it's important to introduce the view menu. The view menus allow you to turn various windows, toolbars, and subsystems on or off as well as open the various Sandbox extended editors and tool dialogs.

While experimenting with views, be aware that if you close a window and want to open it again, this can be done easily using the **View | Open View Pane** menu.



Another important toolset that you will likely want on your layout **is the Rollup Bar**.

The Rollup Bar is similar to the 3ds Command Panels for those already familiar with 3ds. It is a quick menu bar for the majority of the functions available to the editor exposed to the developer in an easily accessible format.

The final important tool you will likely want is the **Console**.

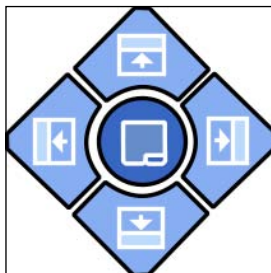
The Console is a direct command-line editor to the CryENGINE 3. This essentially allows access to various advanced functions within the Sandbox editor, including various debug and test profiles.

To start this tutorial, you should have `Sandbox Editor.exe` started.

How to do it...

The first step of customization will be to learn how to scale and move the various windows in Sandbox around:

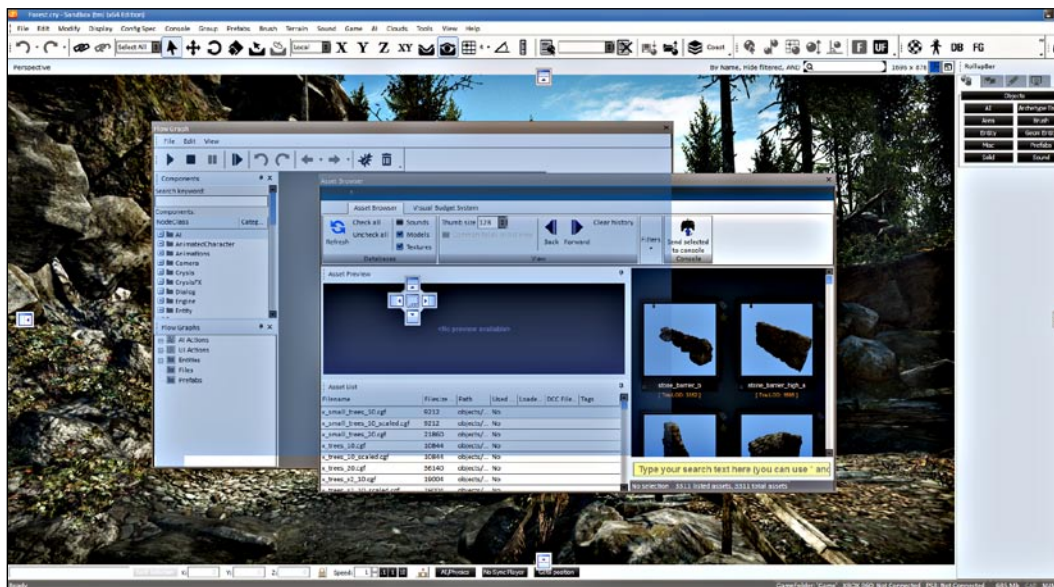
1. Let's first open a new window.
2. Click **VIEW | Open Viewpane | Asset Browser**.
3. This will open up a window containing the asset browser in the centre of your screen.
4. Ignore the contents for now as the asset browser will be explained later on in this recipe; let's resize the window.
5. Move the mouse pointer to the edge of the window, so that it turns into a double-ended black arrow. Click and drag the mouse pointer to scale the window.
6. Now that we have resized the window to our liking, let's use the docking toolbars to anchor the asset browser into the layout.
7. You can see the docking helpers whenever you drag a window over another window, or the Sandbox editor itself.
8. Click and drag the window from the title bar and move it over various docking helpers shown around the main view window.
9. Notice that once you release the mouse button, the window will dock itself into that location.



CryENGINE 3: Getting Started

Now that the window is docked, we should learn how to undock it:

1. Similar to when we docked the window, drag the title bar again and move the selected window away.
2. Notice that the window maintains its original size and shape. You may thus want to resize the window once you have undocked it.
3. Another important interface to master is the ability to dock a window within other windows.
4. Go back to the **View** menu and open another window.
5. For this example, open the **Flow Graph** window.
6. Now, drag the **Asset Browser** window to the **Flow Graph** window.
7. You will observe the docking buttons being displayed again.
8. Use the lower, central button to dock the selected window in the lower half of the **Flow Graph** window.



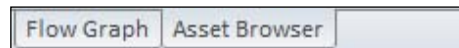
You can also dock windows at the top and sides of other windows using the other docking buttons:

1. To do this drag the title bar of the **Asset Browser** out of the **Flow Graph** window and away to another docking helper within the flow graph window to move it.
2. The final tool that is available to you in customization is **Docking a Window as a Tab in Another Window**.
3. For people using only one monitor, this is almost essential!

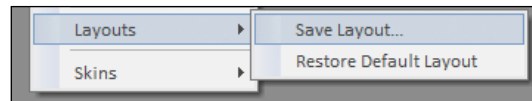
4. You may have noticed previously that when you move a window to another window, a new central, circular button is displayed.



5. This anchor will allow you to place the currently selected window as a tab within another window.
6. Drag the **Asset Browser** to the **Flow Graph** window, and on to the central dock as the tab button.
7. You will notice that there are now two tabs at the bottom of the window, **Flow Graph** and the **Asset Browser**.
8. You can now select each window by clicking on its corresponding tab.



9. Keep in mind that you can undock a window by dragging its tab to another part of the screen.
10. Once you are happy with your layout you can save this layout for easy loading later.
11. To do this, we will access the **Layout Configuration** window on the **Display** menu under **Configure Layout**.
12. Select **Save Layout** from the **Configure Layout** dialog.



13. You will then be presented with an opportunity to name this layout. Type the name of the configuration in this window and click **OK**.

How it works...

The docking helpers work very similar to windows office applications, so any users of those applications may be familiar with this system.

The save layout process creates a folder under the `CryENGINE3/editor` directory called `layouts`.

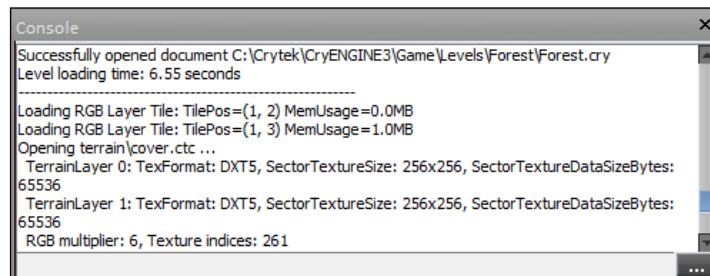
In this folder, it then saves a `.layout` file that is essentially an `.xml` file. This means that it could be edited by hand if required but can also be version controlled, which means multiple presets can be shared across large teams.

The Status Bar

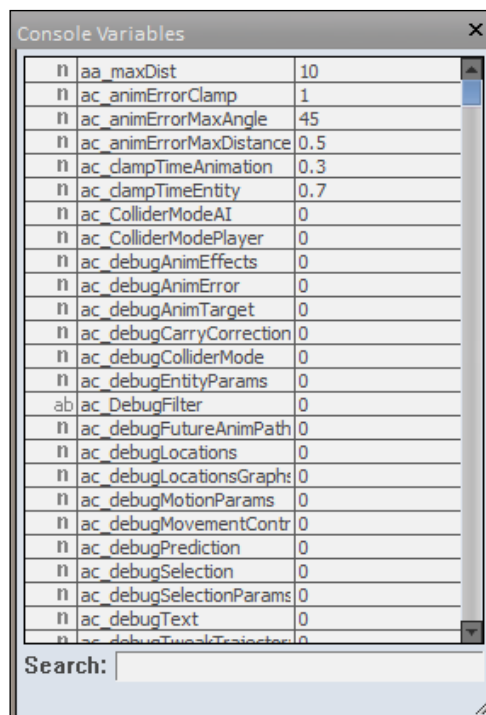
The Status Bar contains translation/rotation/scaling information for selected objects, editor interaction shortcuts, and camera controls.

The Console

The Console in the Sandbox editor is used to input variables. It can be visually toggled on or off by going to the **View Menu** and selecting **View Console** or by pressing the caret key (^) while the **Perspective Viewport** is selected.



In the editor, a full list of console variables can be accessed by double-clicking the input field on the **Console** to open the **Console Variables** window.



Search for variables with partial or complete commands. Information on individual variables can be shown by hovering the mouse over a Console Variable for a couple of seconds in the **Console Variable** window; text will then be displayed as a tool-tip.

The Toolbox

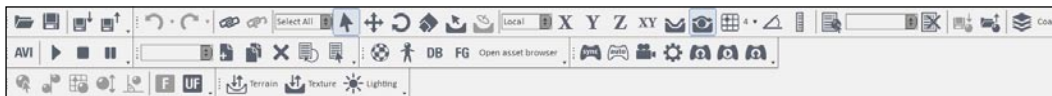
The Toolbox is a set of user-defined tools that contains some example shortcuts to useful editor command lines and different functionality. It can also be added to with user-specific shortcuts and/or console variables.

See also

- ▶ To get right to using some of the interfaces, go to the *Using the Rollup Bar* recipe later in this chapter
- ▶ To learn how to customize toolbars and menus, go to the next recipe

How to customize toolbars and menus

This section will now introduce you to the various toolbars and menus available in the Sandbox. With these toolbars, users can very quickly access many of the features of the Sandbox editor by using simple icons and groups of icons at the top of the interface. These toolbars can be configured to fit the preferences and needs of individual users.



Getting ready

Before adjusting the toolbars, it is important that we explore a brief summary of the default toolbars that are available in Sandbox.



The **Standard Toolbar** contains open, save, hold, and fetch options.

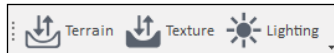


The **EditMode Toolbar** contains **various tools for level editing**. These tools include undo and redo, link and unlink, select all, object movement/scaling, axes and terrain options, as well as object selection, saving, and loading.

The **Object ToolBar** contains tools for object alignment. The icons are go to selected object, align selection, align object to grid, set object(s) height, align object to the surface normal, and fix and unfix selected objects.



With the **Mission ToolBar**, you can select the current mission, duplicate a mission, delete a mission, and reload and edit **mission scripts**.



The **Terrain ToolBar** contains shortcuts to tools within the **Terrain Editor**, the **Terrain Texture Layer** editor, and **Terrain Lighting** dialog.



The **Dialogs ToolBar** contains icons used to access extender editor such as the **Materials Editor**, the **Character Editor**, the **DataBase View**, and the **Flow Graph Editor**.

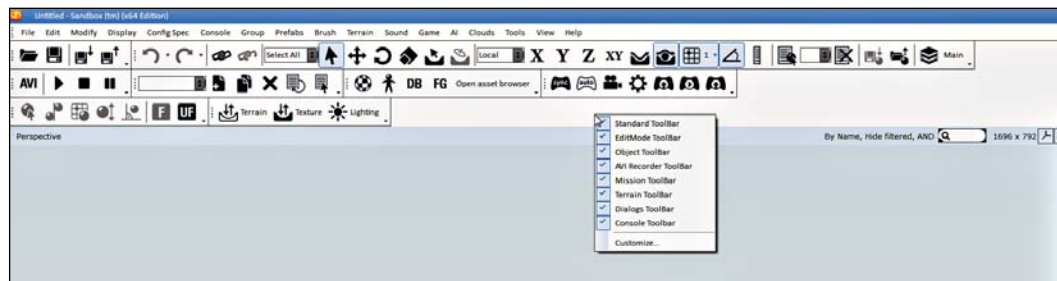


The **Console ToolBar** has options specific to console game development. The buttons include sync data to console, automatically sync data to console, sync camera, options, load current level on console, and launch current level on console.

How to do it...

Now that we know about the default toolbars, let's go ahead and set up our layout:

1. To do this, we will need to access the **ToolBar** settings menu.
2. To access it, right-click anywhere on the **Icon Bar**.

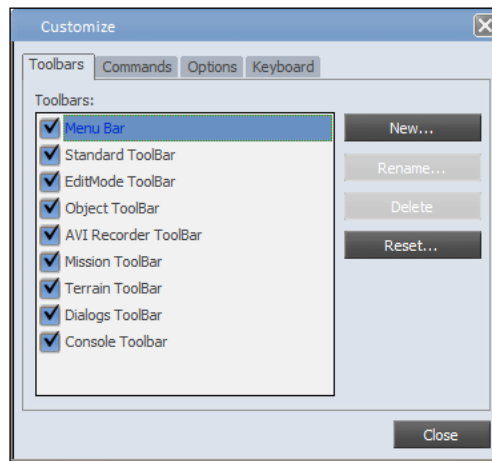


3. This will display the **ToolBar** settings menu.
4. Selecting a toolbar from the list will display it or hide it on the main header.



Toolbars can be arranged horizontally at the top of the editor, vertically on the edges, or completely undocked from the editor.

5. To customize these toolbars and to create new ones simply click on the **Customize** option at the bottom of the **ToolBar** settings menu.
6. The **Customize** dialog box allows users to customize preset toolbars, as well as create custom user toolbars.

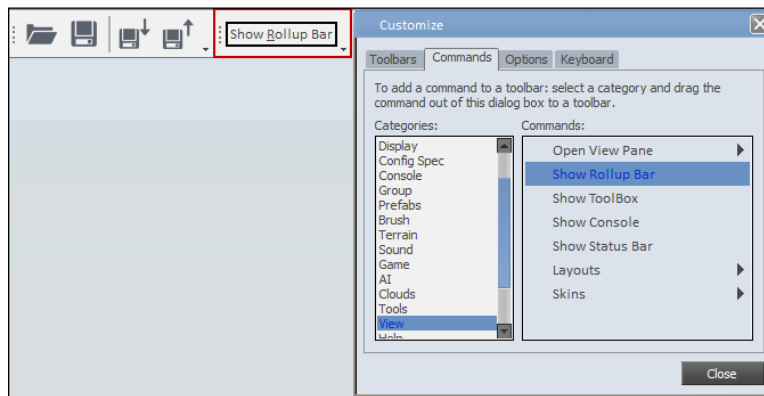


7. The **New** button allows you to create a new custom toolbar. On pressing the button, a prompt will appear requesting a name for the new toolbar.
8. The **Rename** button allows you to rename any of the custom toolbars you have created.
9. The **Delete** button allows you to delete any of the custom toolbars you have created.
10. The **Reset** button returns any changes made to the toolbars back to default.
11. Click on **New** and name it `my_toolbar`.
12. You will then see the toolbar added as an empty container on the interface.



13. We can now add commands to this toolbar for easy and customized access!

14. The **Commands** tab allows you to drag-and-drop any of the icons in the **Commands** box into any toolbar, even custom ones.
15. Go to **View** in the categories options.
16. Click and drag the **Show Rollup Bar** command into your new toolbar.



17. Click **Close** to accept the changes to your custom toolbar.
18. To test its function, click the newly created button and notice that it now toggles the rollup bar on and off.

How it works...

Custom toolbars and overall toolbar configuration is saved in the registry information written by Sandbox to the PC. The toolbars are quite organic in the fact that you can undock and move them anywhere in the Sandbox interface, which, of course, makes it far easier to interface with some of them.

There's more...

You may want to explore some of the other tabs available within the **Customize** interface.

The Options tab

In the **Options** tab, there are some options that allow you to edit the way the interface reacts.

The first option is **Always show full menus**, which as the name suggests always shows the full menu of the currently selected menu.

The next is the **Show full menus after a short delay**, which will, after a short delay, display the full menu even if a skin collapses the menu to only show frequently used items.

The **Reset** menu and toolbars usage data deletes the record of the commands that you've used in the editor, restoring the defaults.

Under the **Other Header**, we have the **Large icons** checkbox. This displays large icons when the editor skin has the choice of using large or small icons.

We also have the **Show screen tips on toolbars** checkbox, which displays screen tips when the mouse is held over toolbar buttons.

There is a subcheckbox to the screen tips, which is to show shortcut keys in screen tips. This shows keyboard shortcuts along with the screen tips.

Finally, we have **Menu animations. This change how menus are displayed. The options include: (System default), Random, Unfold, Slide, Fade, and None.**

Personalized menus and toolbars

The **Keyboard** tab allows the user to assign different shortcuts to certain functions within the editor.

You can browse through different categories using the **Category** drop-down list. Depending on the category selected, different commands will be listed in the commands frame. If the function is already assigned to a key, it will be shown in the **Key Assignments** frame.

To assign a shortcut key, you must have a command selected and then click within the **Press new shortcut key** textbox and that key will then be ready to be assigned to the selected command. To accept the shortcut assignment click the **Assign** button. You can also remove this later by using the **Remove** button.

Using the Rollup Bar

The **Rollup Bar** is one of the most commonly used tools within Sandbox. By default, it is located to the right of the viewport typically along the right edge of the interface. This is where entity parameters, settings, and controls are listed and accessed.

The **Rollup Bar** is split into four very different panels, which are accessed from their corresponding tabs.

The first tab contains the object and entity creation tools for the editor, as well as being the tab that will display all entity-specific information and dialogs.

The second tab has the overall environmental, vegetation, and terrain editing tools. The tools in this tab are used to modify the specific level you currently have loaded in Sandbox.

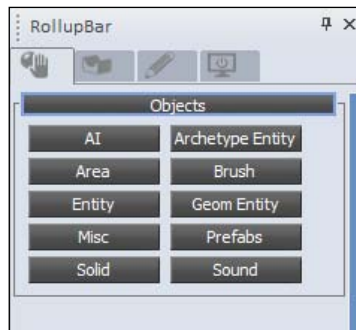
The third tab contains the display options.

The last one is the layer organizational tool.

Getting ready

In this task, we will only use the **Objects** and **Entities** tab, which is the first and default tab within the **Rollup Bar**. To access the majority of scene elements throughout this book, we will use this tab.

It holds interfaces to the various Database libraries and the brush database on your hard drive.



You must have a pre-existing level opened in Sandbox to complete this recipe.

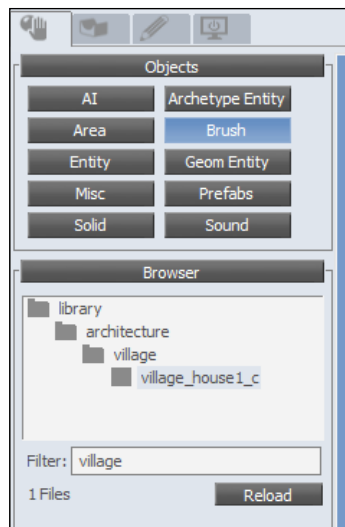
How to do it...

Let's start using the Rollup Bar.

1. In the Rollup Bar, click on the **Brush** button.
2. This will display a browser linked to the `CryENGINE3/Game/Objects` directory of your build.
3. Brushes are compiled geometry containing no extra data other than collision. Typically, most levels are created with brushes as they are simple geometry.

At the bottom of the browser window you will see a dialog box that says **filter:**

1. This is a very useful feature when browsing for specific objects.
2. Type in `village` in the **filter** dialog.
3. Once you hit `enter` the browser will then simplify the contents of the browser to the folder that contained any objects with the name **village** in them.



4. Browse to the default SDK folder `Objects/library/architecture/village/` and notice there is an object called `village_house1_c.cfg`.
5. You can now drag this object out of the brush browser and into your level!

How it works...

Within the **Objects** tab, there are a variety of libraries and object types that support drag-and-drop functionality.

They are separated logically into a few different areas.

There's more...

There are a variety of different sections contained in the Rollup Bar. These sections are further explained later and will be used throughout the course of this book.

The AI section

This section within the Rollup Bar contains AI control objects, which are used to control AI entities and their behaviours in the level. They can define a specific behaviour for an AI with reference to its location or other variables in the level. AI control objects can define navigation paths or an area for the AI on the terrain, including boundaries and forbidden areas. The objects can be used by AI actors to perform specific actions or events, such as animations and changes of behaviour.

The Area section

The Area section contains the area objects, which are used to create three-dimensional zones in the level that can be used to trigger events.

The Entities section

The Entities section contains all the entities, which the player can interact with in some form.

The Misc Objects section

The Misc Objects section includes various tools and functions used during a level's creation such as roads, rivers, and comments.

The Solids section

The Solids tool is used to create simple structures and objects, or placeholders for future art assets. This is one of the best forms of white boxing the engine has available!



Solids can even be exported in an .OBJ format, which is readable by most all of the DCC tools.

The Archetype entity section

The Archetype section allows users to access currently loaded archetype libraries within a given level. An Archetype entity is based on a regular entity and specifies individual parameter values for that regular entity. The main advantage of archetypes is that if the value of an archetype parameter is changed, all instances of that archetype in the level will be updated automatically. Archetype entities are organized into .xml libraries, which can be created in the editor under the Database view.

The Geom entity section

The Geom entity section is a browser similar to the brush browser but instead allows for the placement of a very simple entity that takes its physicalization parameters from its assigned geometry. When objects are placed as Geom entities and it have user-defined properties (discussed later), they become interactive entities with physical values, so they can behave like real-life objects. It is similar to a basic entity, but simpler, more efficient, and has fewer configurable parameters.

The Prefabs section

The Prefabs section contains the currently loaded Prefab libraries for a given level.

Prefabs are groups of objects that can be placed in the level as instances similar to archetype entities. Altering one prefab universally applies the changes to each instance of the prefab object. Any iteration to the prefab is required to be saved to the Prefabs library to ensure they are correctly propagated across an entire series of levels.

The Sound section

This section contains a shortcut to the Entity/Sound section. This allows for the addition of specific sound entities to your level.

See also

- ▶ To use some of these other sections immediately, go to the *Placing enemy AI* recipe in *Chapter 5, Basic AI*
- ▶ To use the Solids section mentioned in this recipe, go to the *Making basic shapes with Solids tool* recipe in *Chapter 3, Basic Level Layout*

Selecting and browsing level objects

To navigate levels is important, but to be able to select and browse your level objects is essential.

To do this, we will be using the **Select Objects** window, which enables you to quickly search for objects, hide, unhide, freeze, and unfreeze objects in a list type view.

Getting ready

To follow the example in this recipe, you should have the `Forest.cry` level loaded in the Sandbox editor.

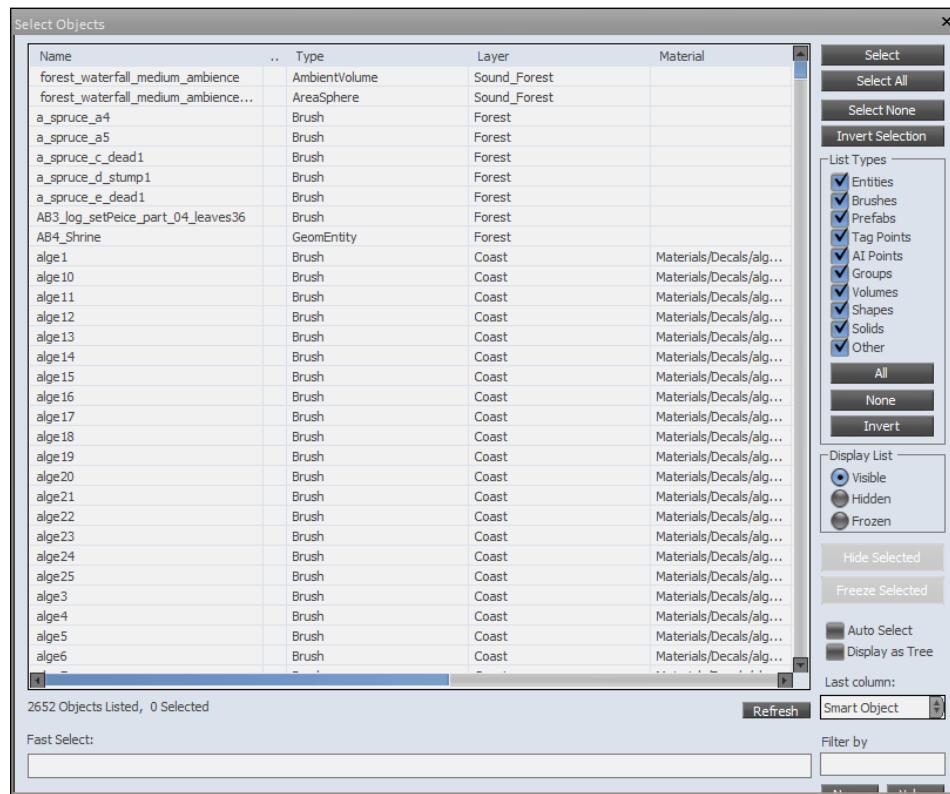
How to do it...

Let's open the **Select Objects** window:

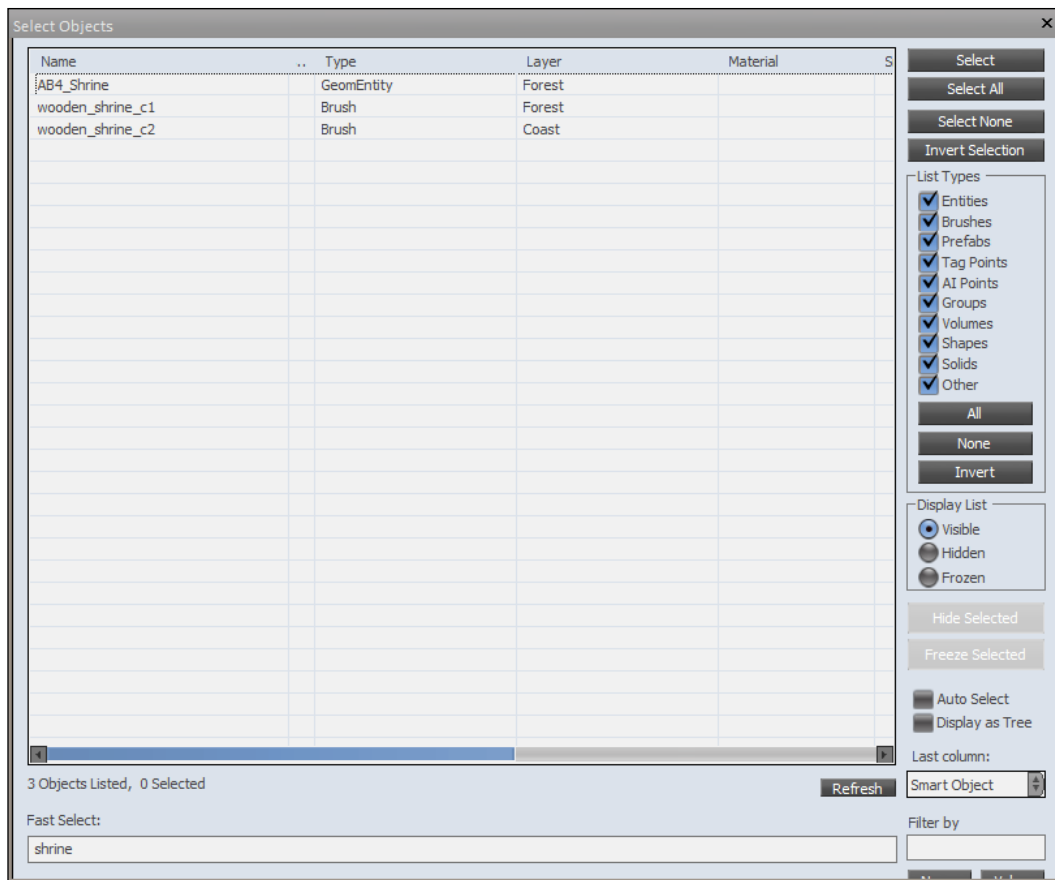
1. You can access the **Select Objects** window by using the **EditMode** toolbar or in the main menu, under **View | Select Object(s)**.

You can also open the **Select Objects** window using the shortcut **Control + T**.

2. Press the heading of the column to sort the level objects based on that particular heading's type. In this case, let's sort by name.



3. You will notice all the objects in the level are listed here.
4. Using the **Fast Select** dialog type in shrine.
5. Use the **Select** button to transfer selections from the object's list to the editor's object selection.



6. You can now combine this with the use of the `goto` object button in the object toolbar.



Remember the `goto` object function moves the editor view to that object's location.

7. Reset the selection using the **Reset** button, which resets the selection in the table and the editor's object selection.

How it works...

The layout of the **Select Objects** window is quite intuitive. It displays a list of the objects in the currently opened level, which can be sorted by the name of the objects, the type, and the layer the object is on or even by the material applied to the object.

You can also use the **Select All** button, which selects all the objects in the table and transfers them to the editor's object selection.

There's more...

There are a good deal of tools available immediately in the **Select Objects** window; you can browse frozen and hidden objects, the type of object, and even show dependencies.

Browsing frozen and hidden objects

You can browse through hidden or frozen objects without changing their corresponding state using the Display list to filter them from the object table.

List types

Enables the display of certain object types (Entities, Brushes, Prefabs); use these filters to display only the data that you need.

Display as a Tree

Indents child objects in Editor Hierarchies visually.

See also

- ▶ To restore all the toolbar's windows and saved settings, you can restore Sandbox's default settings in the next recipe
- ▶ To get right to creating your first level, go to the *Creating a new level* recipe in *Chapter 2, Sandbox Basics*

Restoring the CryENGINE 3 default settings

It is important to know how to reset the settings in Sandbox to their default states.

It can occur that system paths to root builds or folders become corrupted after switching between multiple instances of the CryENGINE 3.

This example will demonstrate how you can restore the default settings by deleting the relevant keys in the Windows Registry editor. This step-by-step process takes you through which keys in the registry should be removed to restore the defaults.

Getting ready

If you are trying to simply restore the Sandbox layout, please try restoring the default layout settings from the **View Menu | Layouts | Restore Default Layout**.



Editing your registry is a very **sensitive operation!** As such, you should back up your registry before starting this tutorial.

After this task you will have restored all Sandbox and resource compiler settings to their default settings.

How to do it...

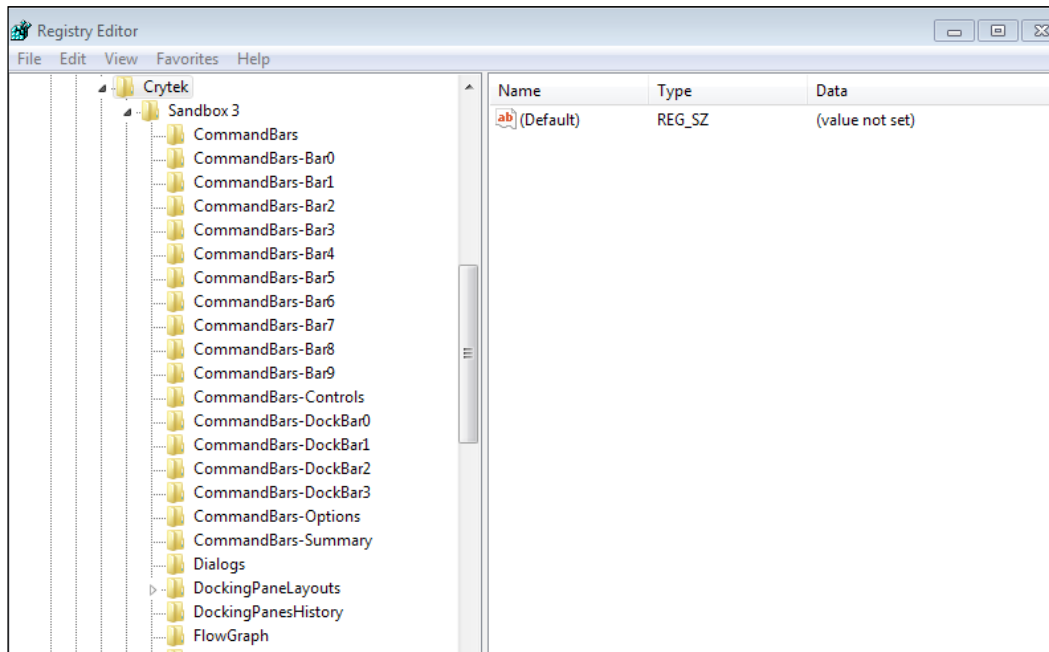
Depending on your Operating System version, the process of opening your PC's registry for edit may be different:

1. To open the Windows Registry editor using Windows XP:

- ☐ Click **Start**
- ☐ Click **Run**
- ☐ Type into the text box, **regedit**
- ☐ Press **Enter**

This will now have opened your Windows Registry editor.

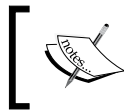
2. In the Registry editor, navigate to HKEY_CURRENT_USER\Software\Crytek\Sandbox 3.



3. When you find the folder **Sandbox 3**, highlight it.
4. Press *Delete*.
5. Now, we can restore the default folder structure automatically by navigating back to the the `Editor.exe` and launching it.

How it works...

Sandbox stores various compulsory values and folders.



It is not recommended to adjust these values manually without having in-depth knowledge of the consequences of the changes.

The registry contains some editor layout data but not all, and because of this, completely restoring the default settings could require removal of other directories, such as the `user` folder.

There's more...

Sometimes it's not required to completely reset the registry to restore some of the settings for Sandbox. You can delete the `user` folder, which is explained further.

Deleting your user folder

The `user` folder might be needed to store user-specific data. Windows can have restrictions on where the user can store files. For example, the program folder might not be writable at all. For that reason, screenshots save game data and other files are typically stored in the `user` folder.

You can simply delete this folder and restart Sandbox for the default settings to be restored.

See also

- Resetting the registry data does not require a re-installation of the software and all required values will be created upon the next launch of Sandbox. Go to the *Starting up the CryENGINE 3 Sandbox* recipe in this chapter to do this

2

Sandbox Basics

In this chapter, we will cover:

- ▶ Creating a new level
- ▶ Generating a procedural terrain
- ▶ Terrain sculpting
- ▶ Setting up the terrain texture
- ▶ Placing the objects in the world
- ▶ Refining the object placement
- ▶ Utilizing the layers for multiple developer collaboration
- ▶ Switching to game mode
- ▶ Saving your level
- ▶ Exporting to an engine
- ▶ Essential game objects
- ▶ Running a map from the Launcher

Introduction

With the **CryENGINE 3 Software Development Kit** installed and ready for use, we can now get into the more exciting bits of utilizing the Sandbox basics.

This chapter will deal with the majority of the tools you will use on a regular basis as well as the essentials for creating new and exciting levels for your project. We will also look at the importance of utilizing the layer system for developers to work simultaneously on the same level. This chapter will also include the use of one of CryENGINE's greatest features of *What you see is what you play* by demonstrating how you may be able to play the level that you have created on demand.

Creating a new level

Before we can do anything with the gameplay of the project that you are creating, we first need a foundation of a new level for the player to stand on. This recipe will cover how to create a new level from scratch.

Getting ready

Before we begin, you must have Sandbox 3 open.

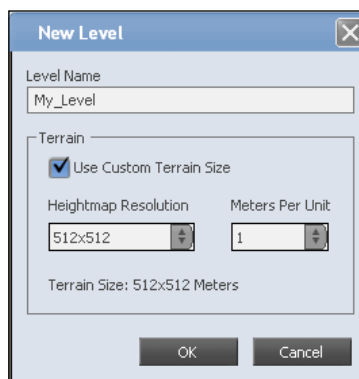
How to do it...

At any point, with Sandbox open, you may create a new level by following these steps:

1. Click **File** (found in the **top-left of the Sandbox's main toolbar**).
2. Click **New**.

From here, you will see a new dialog screen that will prompt you for information on how you want to set up your level. The most important aspect of a level is naming it, as you will *not* be able to create a level **without some sort of proper name for the level's directory and its .cry file**. You may name your level anything you wish, but for the ease of instruction we shall refer to this level as `My_Level`:

1. In the **Level Name** dialog **box**, type in `My_Level`.
2. For the **Terrain** properties, use the following values:
 - ❑ **Use Custom Terrain Size:** True
 - ❑ **Heightmap Resolution:** **512x512**
 - ❑ **Meters Per Unit:** **1**
3. Click **OK**.



Depending on your system specifications, you may find that creating a new level will require anywhere from a few seconds to a couple of minutes. Once finished, the Viewport should display a clear blue sky with the dialog in your console reading the following three lines:

```
Finished synchronous pre-cache of render meshes for 0 CGF's
Finished pre-caching camera position (1024,1024,100) in 0.0 sec
Spawn player for channel 1
```

This means that the new level was created successfully.

How it works...

Let's take a closer look at each of the options used **while creating this new level**.

Using the Terrain option

This option allows the developer to control whether to have any terrain on the level to be manipulated by a heightmap or not. Sometimes terrain can be expensive for levels and if any of your future levels contain only interiors or only placed objects for the player to navigate on, then setting this value to false will be a good choice for you and will save a tremendous amount of memory and aid in the performance of the level later on.

Heightmap resolution

This drop-down controls the resolution of the heightmap and the base size of the play area defined. The settings can range from the smallest resolution (128 x 128) all the way up to the largest supported resolution (8192 x 8192).

Meters per unit

If the Heightmap Resolution is looked at in terms of pixel size, then this dialog box can also be viewed as the *Meters Per Pixel*. This means that each pixel of the heightmap will be represented by these many meters. For example, if a heightmap's resolution has *4 Meters Per Unit (or Pixel)*, then each pixel on the generated heightmap will measure four meters in length and width on the level.

Even though this *Meters Per Unit* can be used to increase the size of your level, it will decrease the fidelity of the heightmap. You will notice that attempting to smoothen out the terrain may be difficult as there will be a wider minimum triangle size set by this value.

Terrain size

This is the resulting size of the level with the **equation of (Heightmap Resolution) x (Meters Per unit)**. Here are some examples of the results you will see (m = meters):

- ▶ (128x128) x 4m = 512x512m
- ▶ (512x512) x 16m = 8192x8192m
- ▶ (1024x1024) x 2m = 2048x2048m

There's more...

If you need to change your unit size after creating the map, you may change it by going into the **Terrain Editor | Modify | Set Unit Size**. This will allow you to change the original *Meters Per Unit* to the size you want it to be.

See also

For more information about additional terrain features refer to the *Generating a procedural terrain* recipe.

Generating a procedural terrain

This recipe deals with the procedural generation of a terrain. Although never good enough for a final product because you will want to fine tune the heightmap to your specifications, these generated terrains are a great starting point for anyone new to creating levels or for anyone who needs to set up a test level with the Sandbox. Different heightmap seeds and a couple of tweaks to the height of the level and you can generate basic mountain ranges or islands quickly that are instantly ready to use.

Getting ready

Have `My_Level` open inside of Sandbox.

How to do it...

Up at the top-middle of the Sandbox main toolbar, you will find a menu selection called **Terrain**. From there you should see a list of options, but for now you will want to click on **Edit Terrain**. This opens the **Terrain Editor** window.

The **Terrain Editor** window has a multitude of options that can be used to manipulate the heightmap in your level. But first we want to set up a basic generated heightmap for us to build a simple map with.

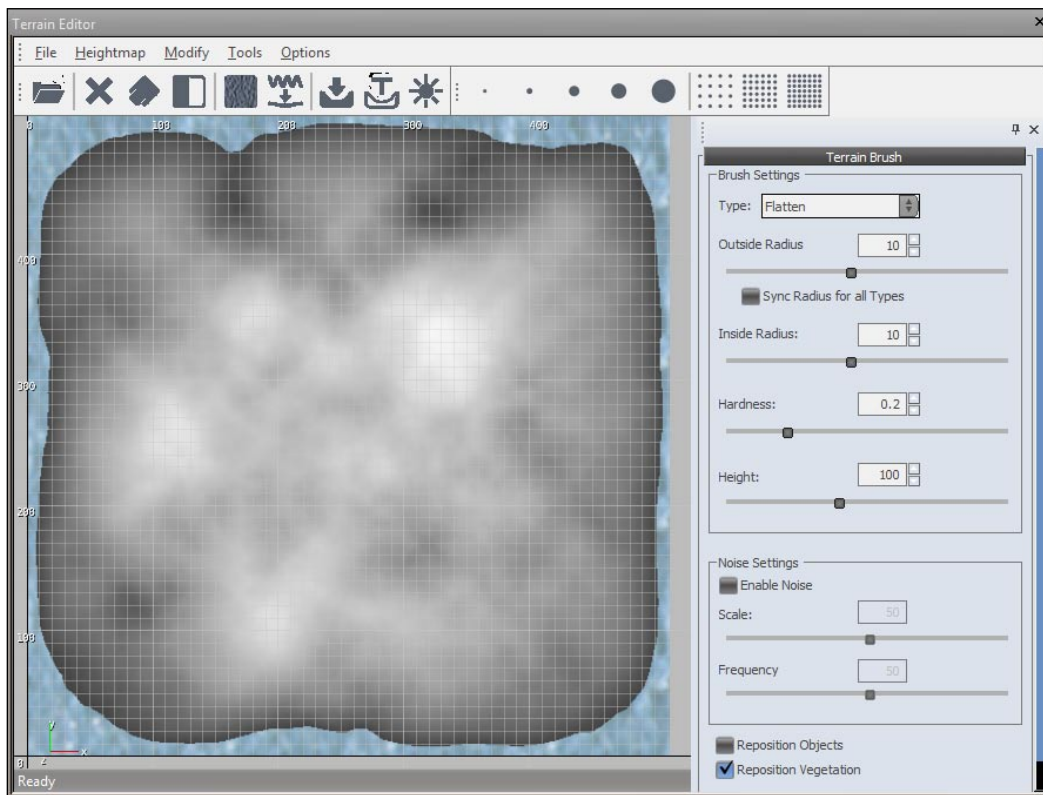
Before we generate anything, we should first set the maximum height of the map to something more manageable. Follow these steps:

1. Click **Modify**.
2. Then click **Set Max Height**.
3. Set your **Max Terrain Height** to **256** (these units are in meters).

Now, we may be able to generate the terrain:

1. Click **Tools**.
2. Then click **Generate Terrain**.
3. Modify the **Variation (Random Base)** to the value of **15**.
4. Click **OK**.

After generating, you should be able to see a heightmap similar to the following screenshot:



This is a good time to generate surface texture (**File | Generate surface texture | OK**), which allows you to see the heightmap with a basic texture in the **Perspective View**.

How it works...

The **Maximum Height** value is important as it governs the maximum height at which you can raise your terrain to. This does not **mean that it is the maximum height of your level entirely**, as you are still able to place objects well above this value. It is also important to note that if you import a grey scale heightmap into CryENGINE then this value will be used as the upper extreme of the heightmap (255,255,255 white) and the lower extreme will always be at 0 (0,0,0 black). Therefore the heightmap will be generated within 0 m height and the maximum height.

Problems such as the following **are a common occurrence**:

- ▶ Tall spikes are everywhere on the map or there are massive mountains and steep slopes:
 - Solution: Reduce the **Maximum Height** to a value that is more suited to the mountains and slopes you want
- ▶ The map is very flat and has **no hills or anything from my heightmap**:
 - Solution: Increase the **Maximum Height** to a value that is suitable for making the hills you **want**

There's more...

Here are some other settings you might choose to use while generating the terrain.

Terrain generation settings

The following are the settings to generate a procedural terrain:

- ▶ **Feature Size**: This value handles the general height manipulations within the seed and the size of each mound **within the seed**. **As the size of the feature depends** greatly on rounded numbers it is easy to end up with a perfectly rounded island, therefore it is best to leave this value at 7 . 0.
- ▶ **Bumpiness / Noise (Fade)**: Basically, **this is a noise filter for the level**. The greater the value, the more noise will appear on the heightmap.
- ▶ **Detail (Passes)**: This **value controls how detailed the slopes will become**. By default, this value is very high to see the individual bumps on the slopes to give a better impression of a rougher surface. Reducing this value will decrease the amount of detail/roughness in the slopes seen.
- ▶ **Variation**: This controls **the seed number used in the overall generation of the Terrain Heightmap**. There are a total of 33 seeds ranging from 0 – 32 to choose from as a starting base for a basic heightmap.
- ▶ **Blurring (Blur Passes)**: This is a **Blur filter**. The higher the amount, the smoother the slopes will be on your heightmap.

- ▶ **Set Water Level:** From the **Terrain Editor** window, you can adjust the water level from **Modify | Set Water Level**. This value changes the base height of the ocean level (in meters).
- ▶ **Make Isle:** This tool allows you to take the heightmap from your level and automatically lowers the border areas around the map to create an island. From the **Terrain Editor** window, select **Modify | Make Isle**.

See also

- ▶ The *Terrain sculpting* recipe
- ▶ The *Setting up the terrain texture* recipe

Terrain sculpting

In this section, we will cover the basics of painting your heightmap by hand. This recipe will teach you how to flatten, raise, lower, and smoothen the heightmap by hand, painting in both the **Terrain Editor** window as well as **Perspective View**.

Using the **Terrain Brush** from the **Terrain Editor** is good for a general high level pass over your level; it is only decent when starting on a fresh level. For more detailed work, many designers use the **Terrain Brush** from **within the Perspective Viewport** to see their results instantly. We will be covering both methods.

Getting ready

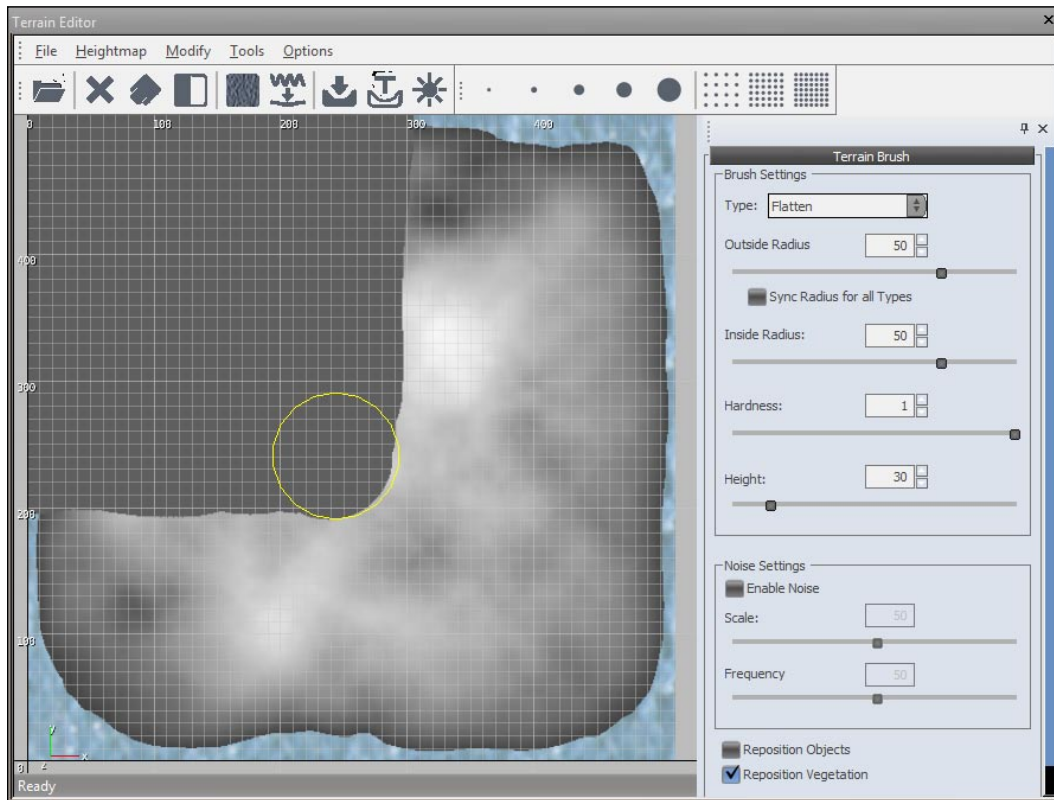
1. Have `My_Level` open inside of Sandbox.
2. Review the *Generating a procedural terrain* recipe to learn about the **Terrain Editor**.
3. Review the *Navigating a level with the Sandbox Camera* recipe to get familiar with the **Perspective View**.
4. Have the **Rollup Bar** available in your Sandbox layout and ready.

How to do it...

On the right-hand side of this window you will see a rollout menu that reads **Terrain Brush**. This menu is the focal point of this recipe and has the same functions in both the **Terrain Editor** and **Perspective View** methods.

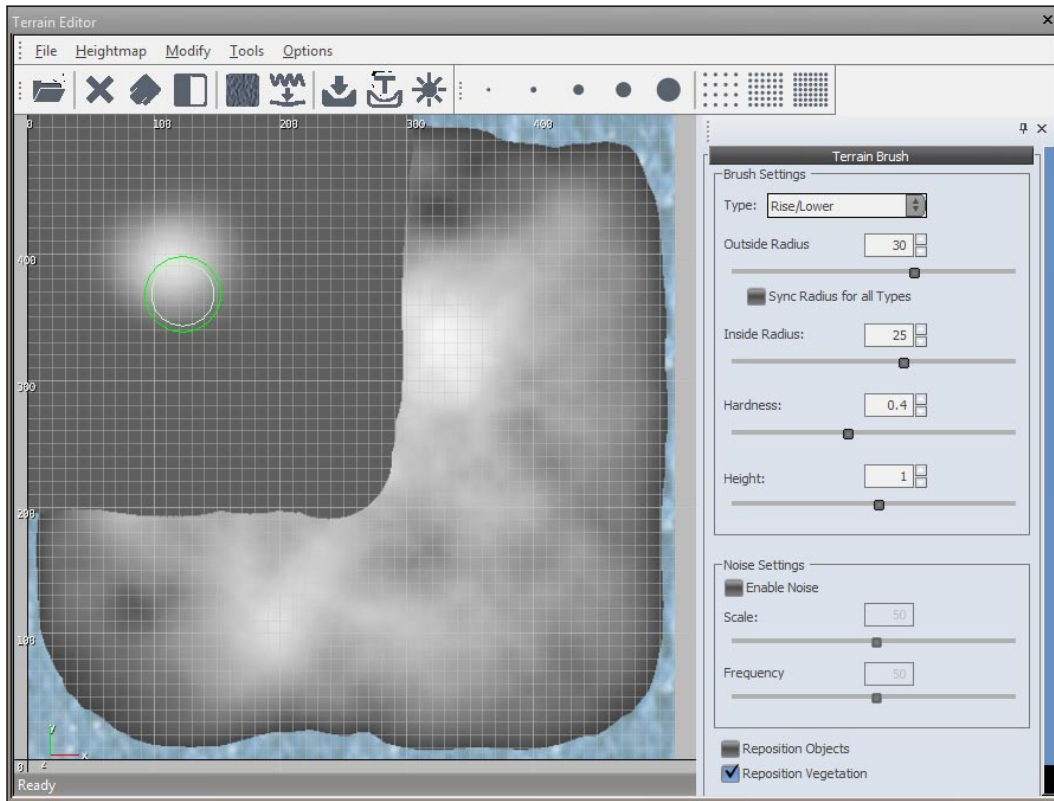
1. To start, we will **flatten a section of the level**. Change the **Type** of brush (drop-down menu to **Flatten**) and set the following parameters:
 - **Outside Radius = 50**
 - **Inside Radius = 50**

- ❑ **Hardness = 1**
 - ❑ **Height = 30**
- 2. Now paint over the north-west corner of the map and flatten that whole quarter of the map.



- 3. Change the **Type** of brush to **Smooth** and set the following parameters:
 - ❑ **Outside Radius = 50**
 - ❑ **Hardness = 0.7**
- 4. Between the flattened area and the rest of the generated heightmap, paint over this area, which **will smoothen the slope between the lower flattened area and the higher up locations**.
- 5. Change the **Type** of brush to **Raise/Lower** and set the following parameters:
 - ❑ **Outside Radius = 30**
 - ❑ **Inside Radius = 25**
 - ❑ **Hardness = 0.4**
 - ❑ **Height = 1**

6. In the **Flattened** area, paint around in the same spot to see the heightmap rise.

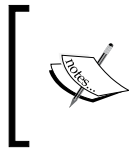


7. Sculpt the terrain in the perspective view. **The Perspective View** method has the very same brushes as demonstrated previously, except you can find where to change them in the **Rollup Bar** under the **Terrain** tab | **Modify** button.

How it works...

Even though each of the brush's functions are self-explanatory based on their name, they share many of the same properties. These properties are **Outside Radius**, **Inside Radius**, **Hardness**, and **Height** (with the exception of Smooth). The following is what each of these properties does:

- ▶ **Outside Radius:** The outer edge of the brush that will feel the least impact from the rest of the brush's parameters.
- ▶ **Inside Radius:** The inner area that will feel the most impact. Depending on **Hardness** and **Height**, there is a fall off between the **Inside Radius** and **Outside Radius**.



To create steeper mounds, hills, or mountains, set the inside and outside radius to be close to each other. To get more shallow and smoother hills, set inside and outside radius further apart.

- ▶ **Hardness:** How hard should the brush be when painting? Basically, if the **Hardness** is set to **1**, then within one click you will have the desired height. If set to **0.01**, then it will take 100 clicks to achieve the same result. (For smooth, this controls how hard it will try to normalize the triangles of the heightmap).
- ▶ **Height:** This controls the desired height you wish the **Flatten** tool to set the terrain to, or controls the step size (meters) of the **Raise/Lower** tool.

There's more...

Here are some other additional settings you may wish to use when sculpting a terrain.

Noise settings

Enabling **Noise** only works with the **Flatten** and **Raise/Lower** brushes. This setting adds a bit of random variation to the heightmap when painting with these brushes.

- ▶ **Scale (%):** How high/low the noise modification will be
- ▶ **Frequency (%):** How often the noise will vary the height along the surface of the terrain

Reposition objects and vegetation

Enabling either of these tickboxes will reposition the Object/Vegetation after the height underneath that Object/Vegetation has changed (this does not work if the object is underneath the terrain).

All vegetation that is affected is under the **Terrain | Vegetation** tab in the **Rollup Bar**.

All objects affected are under the **Objects** tab in the **Rollup Bar**.

See also

- ▶ The *Generating a procedural terrain* recipe
- ▶ The *Navigating a level with the Sandbox Camera* recipe in *Chapter 1, CryENGINE 3: Getting Started*.

Setting up the terrain texture

In this recipe, we will teach you how to set up a new grass-like terrain texture for you to paint on the terrain.

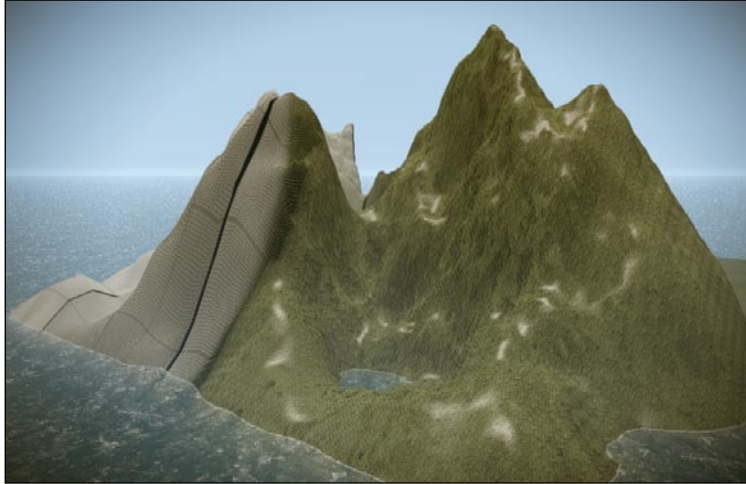
Getting ready

1. Have `My_Level` open inside of Sandbox.
2. Review **the** *Generating a procedural terrain* recipe to learn about the **Terrain Editor**.
3. Review **the** *Navigating a level with the Sandbox Camera* recipe to **ge familia with the Perspective View**.
4. Review **the** *Terrain sculpting* recipe.
5. Have the **Rollup Bar** open and ready.

How to do it...

1. First **ope** the **Terrain Texture Layers** window **foud** in **tb** main **Sandbox Toolbar** | **Terrain** | **Texture**.
2. In this window, create a **new** layer—**Add Layer**.
3. Rename **your NewLayer** to `Grass` (double-click **NewLayer** to rename it).
4. Then change the layer texture **6** **this layer: Change Layer Texture** | **Terrain Folder** | **ground_mud_grey.dds**.
5. **Now** **chang** **tb** detail texture **clik** **a** **tb** material path **fo** this layer, **Materials/** **material_terrain_default** | **Terrain Folder** | **grass_7**. Go back to **Terrain Texture Layers** window | **Assign Material**.
6. Set up your layer to have a **greenib** **colo** **b** it—**RollupBar** | **Terrain** tab | **Layer Painter** | select **Grass**. **Se** **tb** **colo** in **tb** **colo** **bo** to **Red: 191, Green: 215, Blue: 138**.
7. Save the color to your layer and click the **Layer** button above the color box.

8. Paint on your **terrain**.



How it works...

The **Layer Texture** and **Detail Texture** work together in an overlapping fashion to breathe life into the terrain texture that is applied onto the terrain. While the **Layer Texture** works best as an overall brightness randomizer (seen with the *ground_mud_grey* texture) to break up the distant tiling, the **Detail Texture** is the main texture that is seen when closest to the player's view.

It is important to note that the **Detail Texture** should provide the detail only and not so much of the color (the color is handled in the **Layer Texture**). **Grayish textures within the detail** material do not interfere with the colors from the **Layer Texture**. Each of these diffuse textures should be high pass filtered.

There's more...

Here are some additional settings that can be utilized.

Radius and Hardness: Much like **Sculpting Terrain**, you can change the brush size and hardness in the Layer Painter to either a blanket wide area with your Terrain Texture, or just to get those fine details where you need them.

Altitude and Slope: These parameters can be set per layer allowing you to only paint that layer within the Altitude (meters) or Slope (degrees) threshold. This is especially useful for cliff-like layers (for example, 55 to 90).

Filter (Brightness): The Filter slider is an extra brightness pass on the color you have already set. Without needing to change the color over and over, you may change the brightness quickly with this slider.

Tile Resolution: Tile Resolution affects how many terrain tiles are in each terrain sector. The higher the resolution, the higher the amount of tiles used (better layer blending as well as softer transitions in the Layer Texture).

Ranging from 64x64 to 2048x2048, this tiling resolution is an important factor when it comes to optimization as well as high quality terrain. It is recommended that you use a higher value for play areas of your map and low values for terrain sectors that will not have the player in it.

Generating Surface Textures: It is important after doing work with the Terrain Textures on your map to a Generate Surface Texture to bake your textures into your terrain. This creates a compressed version of your painted terrain texture **job into a** .pak file called `terraintexture.pak`. The information stored in this .pak file reduces the total terrain texture size of the map to 1/6 of its original size as well as reducing the amount of drawcalls on the level.

To generate surface textures, go to **File | Generate Surface Texture | Pick resolution** (higher = better quality, lower performance) | (Optional) **High Quality** (does an additional pass over the textures to bake in further detail while keeping the same memory footprint) | **OK**.

See also

- ▶ The *Terrain sculpting* recipe
- ▶ The *Saving your level* recipe

Placing the objects in the world

Placing objects is a simple task; however, basic terrain snapping is not explained to most new developers. It is common to ask *why*, when dragging and dropping an object into the world, they cannot see the object. This section will teach you the easiest ways **to place an object into** your map by using the *Follow Terrain* method.

Getting ready

1. Have `My_Level` open inside of Sandbox (after completing either of the *Terrain sculpting* or *Generating a procedural terrain* recipes).
2. Review the *Navigating a level with the Sandbox Camera* recipe to get **familiar with the Perspective View**.
3. Have the **Rollup Bar** open and ready.
4. Make sure you have the **EditMode ToolBar** open (right-click the top main **ToolBar** and tick **EditMode ToolBar**).

How to do it...

First select the **Follow Terrain** button. Then open the **Objects** tab within the **Rollup Bar**. Now from the **Brushes** browser, select any object you wish to place down (for example, *defaults/box*).

You may either double-click the object, or drag-and-drop it onto the **Perspective View**. Move your mouse anywhere where there is visible terrain and then click once more to confirm the position you wish to place it in.

How it works...

The **Follow Terrain** tool is a simple tool that allows the pivot of the object to match the exact height of the terrain in that location. This is best seen on objects that have a pivot point close to or near the bottom of them.

There's more...

You can **also follow terrain and snap to objects**. This method is very similar to the **Follow Terrain** method, except that this also includes objects when placing or moving your selected object.



This method does not work on non-physicalized objects.

See also

- The *Refining the object placement* recipe

Refining the object placement

After placing the objects in the world with just the *Follow Terrain* or *Snapping to Objects*, you might find that you will need to adjust the position, rotation, or scale of the object. In this recipe, we will show you the basics of how you might be able to do so along with a few hotkey shortcuts to make this process a little faster. This works with any object that is placed in your level, from *Entities* to *Solids*.

Getting ready

1. Have `My_Level` open inside of Sandbox (after completing either the *Terrain sculpting* or *Generating a procedural terrain* recipe).
2. Review the *Navigating a level with the Sandbox Camera* recipe to get familiar with the **Perspective View**.
3. Make sure you have the **EditMode ToolBar** open (right-click on the top main **ToolBar** and tick **EditMode ToolBar**).
4. Place any object in the world.

How to do it...

In this recipe, we will call your object (the one whose location you wish to refine) *Box* for ease of reference.

1. Select **Box**.
2. After selecting **Box**, you should see a three axis widget on it, which represents each axis in 3D space. By default, these axes align to the world:
 - Y = Forward
 - X = Right
 - Z = Up

To move the **Box** in the world space and change its position, proceed with the following steps:

3. Click on the **Select and Move** icon in the **EditMode ToolBar** (1 for the keyboard shortcut).
4. Click on the **X** arrow and drag your **mouse up and down relative to the arrow's** direction.
5. Releasing the mouse button will confirm the location **change**.

You may move objects either on a single axis, or two at once by clicking and dragging on the plane that is adjacent to any two axes: X + Y, X + Z, or Y + Z. To rotate an object, do the following:

1. Select **Box** (if you haven't done so already).
2. Click on the **Select and Rotate** icon in the **EditMode ToolBar** (2 for the keyboard shortcut).
3. Click on the **Z** arrow (it now has a sphere at the end of it) and drag your mouse from side to side to roll the object relative to the axis.
4. Releasing the mouse button will confirm the rotation change.

You cannot rotate an object along multiple axes. To scale an object, do the following:

1. Select **Box** (if you haven't done so already).
2. Click on the **Select and Scale** icon in the **EditMode ToolBar** (3 for the keyboard shortcut).
3. Click on the **CENTER** box and drag your mouse up and down to scale on all three axes at once.
4. Releasing the mouse button will confirm the scale change.

It is possible to scale on just one axis or two axes; however, this is *highly* discouraged as **Non-Uniform Scaling** will result in broken physical meshes for that object. If you require an object to be scaled up, we recommend you only scale uniformly on all three axes!

There's more...

Here are some additional ways to manipulate objects within the world.

Local position and rotation

To make position or rotation refinement a bit easier, you might want to try changing how the widget will position or rotate your object by changing it to align itself relative to the object's pivot. To do this, there is a drop-down menu in the **EditMode ToolBar** that will have the option to select **Local**. This is called **Local Direction**.

This setup might help to position your object after you have rotated it.

Grid and angle snaps

To aid in positioning of non-organic objects, such as buildings or roads, you may wish to turn on the **Snap to Grid** option. Turning this feature on will allow you to move the object on a grid (currently relative to its location). To change the grid spacing, click the drop-down arrow next to the number to change the spacing (grid spacing is in meters).

Angle Snaps is found immediately to the right of the **Grid Snaps**. Turning this feature on will allow you to rotate an object by every five degrees.

Ctrl + Shift + Click

Even though it is a *Hotkey*, to many developers this hotkey is extremely handy for initial placement of objects. It allows you to move the object quickly to any point on any physical surface relative to your *Perspective View*.

See also

- The *Placing the objects in the world* recipe

Utilizing the layers for multiple developer collaboration

A common question that is usually asked about the CryENGINE is how does one developer work on the same level as another at the same time. The answer is—*Layers*. In this recipe, we will show you how you may be able to utilize the layer system for not only your own organization, but to set up external layers for other developers to work on in parallel.

Getting ready

1. Have `My_Level` open inside of Sandbox (after completing either the *Terrain sculpting* or *Generating a procedural terrain* recipe).
2. Review the *Navigating a level with the Sandbox Camera* to get familiar with the **Perspective View**.
3. Review the *Using the Rollup Bar* recipe.
4. Have the **Rollup Bar** open and **ready**.
5. Review the *Placing the objects in the world* (place at least two objects) recipe.

How to do it...

1. For this recipe, we will assume that you have your own repository for your project or some **means to send your work to others in your team**.
2. First, start by placing down two objects on the map. For the sake of the recipe, we shall **refer to them as Box1 and Box2**. After you've placed both boxes, open the **Rollup Bar** and bring up the **Layers** tab.
3. Create a new layer by clicking the **New Layer** button (paper with a + symbol).
4. A **New Layer** dialog box will appear. Give it the following parameters:
 - ❑ **Name** = `ActionBubble_01`
 - ❑ **Visible** = **True**
 - ❑ **External** = **True**
 - ❑ **Frozen** = **False**
 - ❑ **Export To Game** = **True**

5. Now select **Box1** and open the **Objects** tab within the **Rollup Bar**.
6. From here you will see in the main rollup of this object with values such as – **Name**, **Helper Size**, **MTL**, and **Minimal Spec**. But also in this rollup you will see a button for layers (it should be labelled as **Main**). Clicking on that button will show you a list of all other available layers.
7. Clicking again on another layer that is not highlighted will move this object to that layer (do this now by clicking on **ActionBubble_01**).
8. Now save your level by clicking—**File | Save**.



There is important information about saving your level in that recipe. Please make a note to review it.

Now in your `build` folder, go to the following location: `... \Game\Levels\My_Level`. From here you will notice a new folder called `Layers`. Inside that folder, you will see `ActionBubble_01.lyr`.

This layer shall be the layer that your other developers will work on. In order for them to be able to do so, you must first commit `My_Level.cry` and the `Layers` folder to your repository (it is easiest to commit the entire folder).

After doing so, you may now have your other developer make changes to that layer by moving **Box1** to another location. Then have **them save the map**.

Have them commit *only* the `ActionBubble_01.lyr` to the repository. Once you have retrieved it from the updated repository, you will notice that **Box1** will have moved after you have re-opened `My_Level.cry` in the Editor with the latest layer.

How it works...

External layers are the key to this whole process. Once a `.cry` file has been saved to reference an external layer, it will access the data inside of those layers upon loading the level in Sandbox.

It is good practice to assign a **Map owner** who will take care of the `.cry` file. As this is the master file, only one person should be in charge of maintaining it by creating new layers if necessary.

There's more...

Here is a list of limitations of what external layers cannot hold.

External layer limitations

Even though any entity/object you place in your level can be placed into external layers, it is important to note that there are some items that cannot be placed inside of these layers. Here is a list of the common items that are solely owned by the `.cry` file:

- ▶ Terrain
 - Heightmap
 - Unit Size
 - Max Terrain Height
 - Holes
 - Textures
- ▶ Vegetation
- ▶ Environment Settings (**unless forced through Game Logic**)
 - Ocean Height
- ▶ Time of Day Settings (unless forced through Game Logic)
- ▶ Baked AI Markup (The owner of **the** `.cry` file must regenerate AI if new markup is created on external layers)
- ▶ Minimap Markers

See also

- ▶ The *Saving your level* recipe
- ▶ The *Exporting to an engine* recipe
- ▶ The *Essential game objects* recipe

Switching to game mode

CryENGINE prides itself on the saying *What you see is what you play*, and the switching to game mode is a testament to this.

It goes without saying that testing your work often is the key to a successful project, and the quick easy use of switching to **Game Mode** within the **Editor** allows you to test at will without the need to close Sandbox.

Even though the Editor is built with this feature to allow Developers to check their work, this is still only an emulation. This means that there are several special editor rules and debug options available in this mode, which isn't fully representative of what the Player might see in the **Pure Game**. Even though it is an excellent idea to test often by switching to Game Mode, it is also important to do any final testing in the **Launcher** on your target platform to have a proper **Pure Game** experience.

In this recipe, we will cover the simple, but highly important use of switching to game mode. This function allows you to jump into your level instantly, and allows you to test what the player will see, hear, and feel within the location you are at with your *Perspective Camera*.

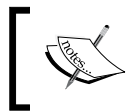
Getting ready

Have `My_Level` open inside of Sandbox (after completing either the *Terrain sculpting* or *Generating a procedural terrain* recipe).

Review the *Navigating a level with the Sandbox Camera* to get familiar with the *Perspective View*.

How to do it...

In the Sandbox main toolbar, find **Game | Switch to Game** or click on *Ctrl + G*. It's that simple.



Game Logic can change the player's initial spawning location. If this logic is enabled, then you will be forced to spawn at this location as well when entering game mode.

See also

- ▶ The *Essential game objects* recipe
- ▶ The *Running a map from the Launcher* recipe

Saving your level

On the surface, saving your level seems like a simple process; however, there are a few important functionalities that you should be aware of when saving your level. This recipe will show you how to do a basic saving, copying/moving your level (do *not* use **Save As**), and auto backups.

Getting ready

Have any level open inside Sandbox.

Open Windows Explorer to the following location: `... \CryENGINE_Build\Game\Levels`.

How to do it...

To save a level, go to **File | Save**.



Ctrl + S does not work by default. You will need to set up a shortcut key yourself for that.

When copying/moving levels:

1. Initially, save your level to its default location.
2. Go to the level's location in Windows Explorer (...\\Game\\Levels).
3. Using Windows commands copy/cut the entire `My_Level` folder.
4. Paste the level into the location you want.
5. If necessary, rename both the `My_Level` folder and `My_Level.cry` to something you prefer.

We recommend that you *never* perform a **Save As** for your level. Unfortunately, there are several dependencies (`Level.pak`, `TerrainTexture.pak`, `Layers`, `Minimap` images, and so on) that do not get moved or relink themselves when a **Save As** is performed. They are all dependent on the folder in which they are placed. If you do this, you will see several anomalies within your level (broken textures, bugs in game logic, missing assets).

When creating a backup, remember that by default, Sandbox creates `My_Level.bak` files after the second/third time your level is saved. These `.bak` files are essentially the previous saves of your `.cry` file.

If, for whatever reason, you need to revert back to a previous save of your level, you may delete your `My_Level.cry` file and rename `My_Level.bak#` to `My_Level.cry` and reopen your level inside of Sandbox.

Also, by default, Sandbox creates up to two previous revisions (`My_Level.bak` and `My_Level.bak2`).

How it works...

By default, all levels are saved in their own folder inside of ...\\Game\\Levels by their level name. For example, `My_Level` will have its own folder with the level's folder. Once saved, each level will also contain a `*.cry` file, with the `*` being the name of the level as well. Each of these `.cry` files hold all the relevant information that is required to make up your level (much like a blueprint for your level).

There's more...

Depending on your work style, you might want to enable the Auto Backup system. This will automatically save your work every X minute interval with the name of your choice (**Autobackup** by default) into your level's folder. These Auto Backups will save continuously as long as your editor is open or you turn this feature off.

To access this feature, open **Sandbox | Tools | Preferences | General/Files | Auto Backup**.

Exporting to an engine

Even though saving your level will create a `.cry` file for you, it is extremely important to note that a `.cry` alone will not work in the **Launcher**. For that you will need to perform this recipe, which will create the required files to run in **Pure Gamemode**.

Getting ready

Have `My_Level` open inside Sandbox.

How to do it...

Go to **File | Export to Engine** or click on `Ctrl + E`.

How it works...

This crucial step is required to convert all the *blueprint's* setup in your `.cry` file into a `level.pak` inside of the `My_Level` folder to be used in **Pure Gamemode**. This `level.pak` is basically a cache of data for the game to use, which houses all the baked information about Game Logic, AI Markup, Particle list, Brush list, and so on.

There's more...

Here is some useful information that you should know about `.pak` files.

Opening .pak files

It is possible to open these `.pak` files and see the information stored inside them by using third-party compression software, such as WinZIP and WinRAR. However, we do not recommend doing this unless you are familiar with handling the files you may wish to manually change. If you do change the files that are inside of these `.pak` files, you run the risk of damaging the `.pak` file and breaking your level for the Launcher.

Corrupted .pak files should be deleted and re-exported

There is no need to panic if, for whatever reason, any of your .pak files become corrupted and unusable in **Pure Gamemode**. You can always delete them and generate a new one from your `My_Level.cry` by repeating this recipe.

Essential game objects

Even though we covered how to export your level to be used in **Pure Gamemode**, you may find that when you launch your level, you are not in the correct location in the map. This quick recipe will show you how to create a Spawn point for **Pure Gamemode** and also remind you the importance of having a physical surface to spawn on.

Getting ready

1. Have `My_Level` open inside of Sandbox (after completing either the *Terrain sculpting* or *Generating a procedural terrain* recipe).
2. Review the *Navigating a level with the Sandbox Camera* to get familiar with the **Perspective View**.
3. Review the *Placing the objects in the world* and *Refining object placement* recipes.
4. Have the **Rollup Bar** available in your Sandbox layout and ready.

How to do it...

Unless your game supports some sort of Player flying mechanic, we recommend you choose a location where the Player can stand when he initially spawns within your level. Any physical surface (Floor, Roof, Terrain, Rock, and so on) will do.

To place a *Spawn Point* follow these steps:

1. Go to your **Rollup Bar**.
2. Open the **Objects** tab and click the **Entity** button.
3. Open the **Others** folder in the Entity List.
4. Place *Spawn Point* in the location you wish to spawn the player.

How it works...

This initial spawn point will automatically work as the main spawn point for your map in **Pure Gamemode** (as long as you keep to the default `SinglePlayer.lua` gamerules). If you jump into your map from the **Launcher** now, you will be spawned at this location.

Any additional spawn points will not function unless there is some game logic hooked into their functionality.

See also

- ▶ The *Running a map from the Launcher* recipe

Running a map from the Launcher

The final recipe in this chapter will cover how you may access the **Pure Gamemode** version of `My_Level` from the **Launcher**.

Getting ready

You must have a map with a working spawn point (otherwise you will spawn at 0,0,0), that has been exported from the engine with a **functional** `level.pak`.

The level must also be inside your `Build` folder.

How to do it...

1. From Windows Explorer, open either `...\Bin32` or `...\Bin64`.
2. Launch the executable `Launcher.exe`.
3. Open the console (`~`).
4. Type the following: `map My_level`.
5. Then press *Enter*.

This will load the map as well as spawn you in the spawn point that was provided.

See also

- ▶ The *Starting up the CryENGINE 3 Sandbox* recipe
- ▶ The *Essential game objects* recipe
- ▶ The *Saving your level* recipe
- ▶ The *Exporting to an engine* recipe

3

Basic Level Layout

In this chapter, we will cover:

- ▶ Making basic shapes with the Solids tool
- ▶ Combining the solids to make white box assets
- ▶ Grouping the objects
- ▶ Utilizing the Geom entities instead of Brushes
- ▶ Road construction
- ▶ Painting vegetation
- ▶ Breaking up tiling with Decals
- ▶ Making caves with Voxels
- ▶ Creating Prefabs to store in the external libraries

Introduction

Now that we've mastered the CryENGINE 3 Sandbox basics, we can move on to adding particular objects and utilizing certain tools that will give the level the feel of a real level. In this chapter, we will look at how you will be able to utilize these objects and tools to visually make up a level that looks like something more than just a test map.

Making basic shapes with the Solids tool

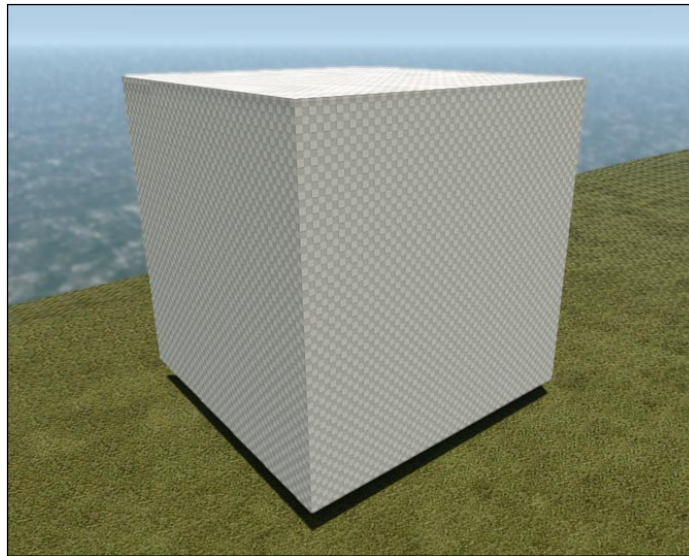
An essential tool for any level designer is the Solids tool. This tool can be regarded as a very simplistic shape creator. Ideally, you will eventually want to get an artist to build the proper asset for you in other external 3D applications. However, by using the Solids tool, you will still be able to create simplistic white box assets to get both the object's volume footprint and dimensions.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`

How to do it...

1. In the RollupBar, click on the **Solid** button.
2. With the **Follow Terrain** enabled and **Grid snap** set to **1** meter, click and drag anywhere on your terrain.
3. Create a 30x30x30 meter cube by following the measured size displayed on each axis.



How it works...

The Solids tool is a very basic geometry creation tool within the editor. It allows you to create a multitude of different shapes that all have their own physical proxies tied to them. You can use these shapes to form complicated objects and merge them together, allowing you to export them with their size intact for an artist to later create into an asset; however, this step will be explained later.

There's more...

Here are some additional properties you may add to the solids when creating them.

Other shapes—cone, sphere, cylinder

When creating a new solid, boxes aren't the only shape. You will be able to create shapes such as cones, spheres, and cylinders.

Number of sides—only for cone, sphere, cylinder

The `Num Sides` parameter is used to define how many sides you want your solid to have. This parameter only changes the initial number of sides the shape will have for cones, spheres, and cylinders.

See also

- ▶ The *Editing and Merging Solids* recipe

Editing and merging solids

The next step is modifying your solids to what you want them to portray. We will be exploring the possibilities of both editing the solids as well as setting them up to be exported for an external 3D application, such as 3DS Max.

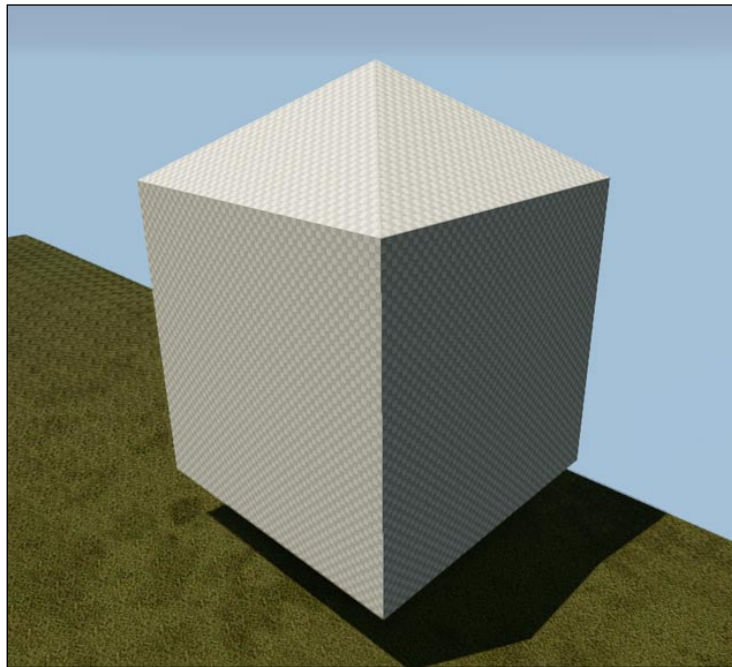
Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Create a *Box Solid* with dimensions of `30m x 30m x 30m`

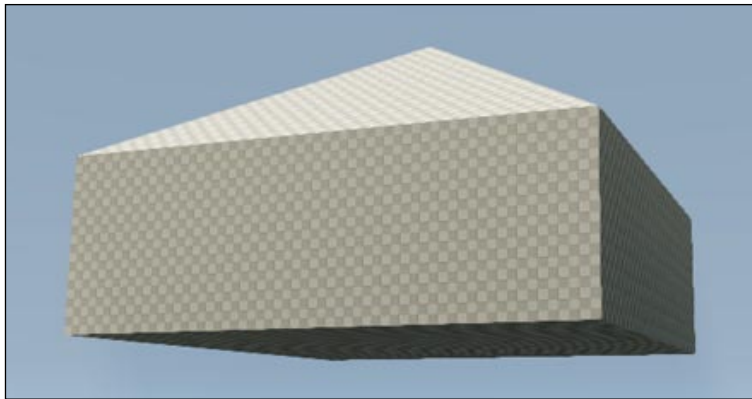
How to do it...

For those who are familiar with DCC tools, such as 3DS Max or Maya, *Editing Mode* should look very familiar. It allows you to modify the shape of the solid by changing the location of vertices, edges, faces, and polygons. For those who are unfamiliar with it, we will now demonstrate how to turn this *Cube* into a very simplistic house-like structure:

1. Start by selecting your object and clicking **Editing Mode**.
2. Change the **Selection Type** to **Polygon**.
3. Select the top side of the cube.
4. Under the **Sub Object Edit** roll out, select **Face [Split]**.
5. Change your **Selection Type** from **Polygon** to **Vertex**.
6. Select the top most **Vertex** (it should be the newly created one in the centre of the face that was split) and drag it up a little on the **Z Axis**, so that the total height of the Solid is now **40** meters.



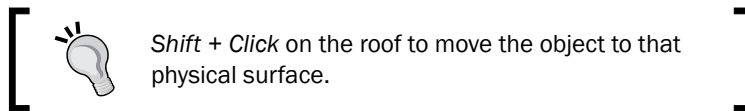
7. Change your **Selection Type** back to **Polygon**.
8. Select the bottom face and move it up to the point where the total height of the *solid* is now **20** meters:



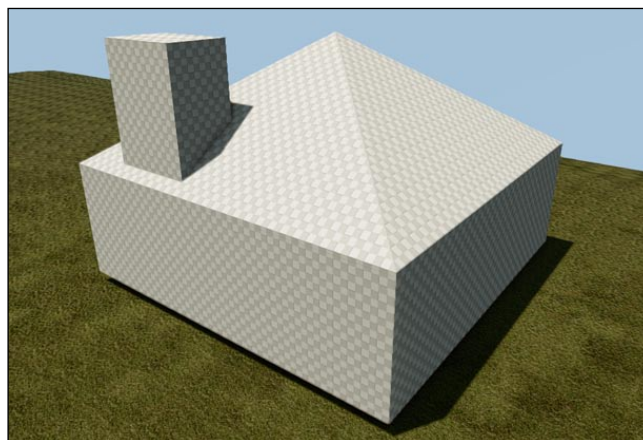
9. Place the *solid* back onto the ground.

You may copy solids and place them anywhere on your level as they too follow the same basic object placement methods used in the Editor. You may also create additional solids that clip through the faces of other solids without interfering with the physics of either solid. This allows you to stack multiple basic shapes of the solids to form the very basic shapes required for an object, or form elaborate structures. Now, let's create one by making a very basic chimney for the *house* we created:

1. Create a new **Box Solid** with the dimensions of **5x7x10** meters.
2. With the new Box selected, move it to any place on the roof you'd like.



Shift + Click on the roof to move the object to that physical surface.



Even after creating multiple solids and perhaps just stacking them on top of each other for a shape that you want, it won't combine them into a single mesh for the artists to look at within their DCC. To do this, first we need to merge the objects into a single mesh:

1. Select both the solids.
2. In the **Solid Parameters** rollout, click the **Misc [Merge]** button.

There's more...

These are some ways you can export your *solid* to directly port over to a DCC toolkit.

Exporting the selected geometry to .OBJ

To export the selected geometry placed inside the Editor, click on **File | Export selected geometry to .OBJ**. This allows you to export the newly merged solid for an artist to take and remake it into a real game asset, all the while keeping to the dimensions you've made from the Solids tool.

Resetting the XForm

If any of your solids were rotated/scaled, you may wish to reset the XForm before exporting the Geometry. This makes sure that all of the transformations on the solid are zeroed out and reset to the world coordinates. If this step is not taken before the Export, you may find that the object will be misaligned when opened with an external DCC.

See also

- ▶ The *Making basic shapes with the Solids tool* recipe

Grouping the objects

This is a simple but extremely useful tool for any designer who wants to move multiple objects at the same time without always having to select all of them. This is also another way to keep objects aligned to one another when rotating.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Place two objects of any kind near each other (Brush, solid, and so on)

How to do it...

1. Select both the **Objects**.
2. On the **Top Main** toolbar, click **Group | Group**.
3. Give it the name `MyFirstGroup`.

How it works...

The **Group** tool allows you to group objects together and move them in unison as if they were one object. You can also clone groups, which makes it easier to populate an area with similar objects.

There's more...

Without ungrouping the objects, it's still possible to change any of the properties of an object within that group. Simply **Open** the group and select the object you wish to manipulate inside.

To close the group, select the group and click the **Close** button.

See also

- ▶ The *Creating Prefabs to store in the external libraries* recipe

Utilizing the Geom entities instead of brushes

In this recipe, we will show you how to place a **Geom** entity in your level for later use in the **FlowGraph**, which will be handled in a different recipe.

Getting ready

- ▶ Before we begin, you must have **Sandbox 3** open
- ▶ Then open `My_Level.cry`

How to do it...

1. From the **RollupBar**, click **Geom Entity**.
2. Place down the **default/box** object.
3. Right-click the **box** object and select **Create Flow Graph**.
4. Name your **FlowGraph** `My_Box_FG`.

How it works...

Using the Geom entities instead of standard brushes is usually reserved for when a designer wishes to perform some game logic or functions on an object. Anything from FlowGraph Scripting to Trackview animations requires the use of entities and cannot manipulate brushes.

Geom entities are very basic entities that can be of any object type (.cgf, .cga, .chr), whereas brushes are usually reserved for static meshes (.cgf). Even though Geom entities have very few properties associated with them, they are still entities within the world. And having too many entities in the world tends to add to a performance drop within the game. This is why static objects should always be brushes instead of Geom entities.

See also

- ▶ The *Creating a new Trackview Sequence* recipe
- ▶ The *Game Logic Chapter* recipe

Road construction

In this recipe, we will explore the basic uses of the Road tool and how to paint down paths that are easy for the player to recognize.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`

How to do it...

1. From the RollupBar, open the **Objects** tab.
2. Select the **Misc** button.
3. From the **Object Type**, select **Road**.
4. Make sure you are set to **Align to Terrain**.
5. Click down three points in roughly a straight line.
6. Double-click for the fourth point to finalize the road.
7. Open the **Material Editor (M)**.
8. Assign the material **materials | Terrain | roads | road_asphalt01** to the road.



How it works...

The Road tool is a spline that projects a repeating texture along the terrain surface.

There's more...

The following are some of the definitions to the parameters for the road as well as how you can edit the shape of the road.

Road parameters

- ▶ **Width:** This is the width of the road in meters.
- ▶ **StepSize:** This affects the tiling amount on the road. It is used for both the texture and the curves on the road.
- ▶ **TileLength:** This affects the stretching amount in just the road texture.
- ▶ **SortPriority:** This determines the priority of which projected texture should be displayed on top (Decals match this priority range).

Shape editing

You may make modifications to any pre-existing road by selecting it and turning on **Edit** from the **Shape Editing** menu item. The Road tool's spline is affected by where the points are situated between each other. From the **X** and **Y** locations of the points, curves are automatically created and projected against the ground. The **Z** axis makes little difference in the way the road is projected onto the terrain; however, it is still a good practice to maintain the road spline above and close to the terrain surface.

- ▶ **Adding Points:** *Ctrl* + click to add additional points.
- ▶ **Angle:** You may change the angle at which the projection of the road is casted. The angles range from -25 to +25.
- ▶ **Individual Point Width:** Turning off the **Default width option** will enable you to adjust the width of the individual points on the road.

Align Height Map

This tool is extremely useful for any designer who wishes to quickly align his/her height map to the Z locations on the road. This is also an excellent method for making ramps.

See also

- ▶ The *Terrain Sculpting* recipe in *Chapter 2, Sandbox Basics*.

Painting vegetation

In this recipe, we will be showing you how you may bring some life to the your level's environment by painting down vegetation objects, such as trees.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`

How to do it...

1. From the RollupBar, open the **Terrain** tab.
2. Click the **Vegetation** button.



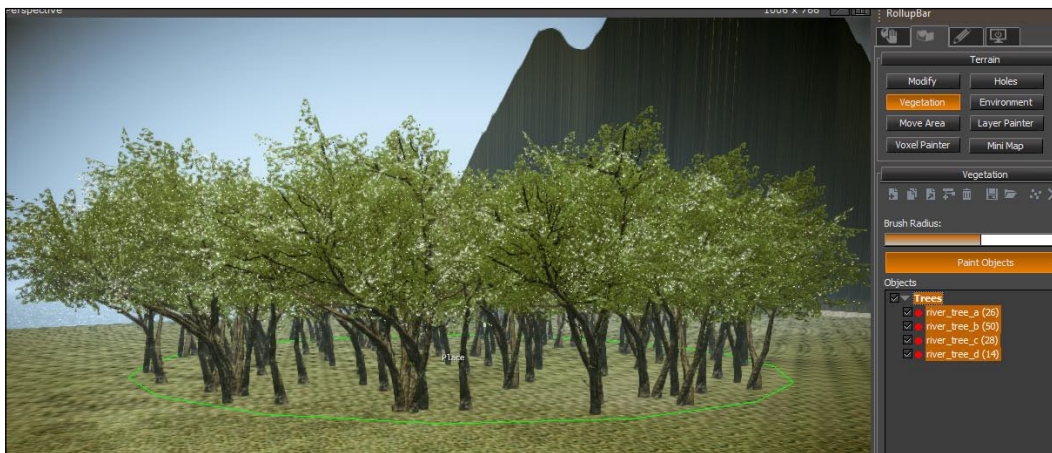
Before we can paint the vegetation down, we must first add in some objects that we want to use as vegetation. The first vegetation we will place down will be some trees.

3. Click on the **Add Vegetation Category**.
4. Name the new category as **Trees**.
5. Select the newly created **Trees** category.
6. Click the **Add** button (page with the +).
7. Navigate to **.../Objects/Natural/trees/river_tree**.
8. **Ctrl** + click on **river_tree_a**, **b**, **c**, and **d.cgf**.
9. Click **Open**.

All of the highlighted trees have now been placed in this category. You can now start painting with a few of the selected trees, or select the entire category to paint everything within. We shall demonstrate this by selecting the entire category:

1. Select the **Trees** category.
2. Set the **Brush** radius to **50%**.
3. Click **Paint Objects**.
4. Click somewhere on your terrain.

You have now painted a small grove.



How it works...

The vegetation painter is a lot like placing brushes into a level. The major difference between the two is that the vegetation paint allows you to paint multiple objects at a time while following ONLY the terrain, whereas brushes allow you to place one object at a time, but in any location/direction you desire. It is possible to use objects other than just trees or grass for the vegetation painter, but keep in mind that you will not have the same control as with brushes.

There's more...

Here are the definitions to the vegetation parameters.

Vegetation parameters

- ▶ **Size:** This is the ratio of how big the object is from its original size (1 = default object size).
- ▶ **SizeVar:** This is a random variation in size that is painted down after taking the *Size* parameter. For example, Size = 2, SizeVar = 0.3. The final size of painted objects vary between 1.7 to 2.3.
- ▶ **RandomRotation:** This randomizes the rotation of the painted objects that are placed.
- ▶ **AlignTo Terrain:** This aligns the objects to the Normal of the terrain surface.
- ▶ **UseTerrainColor:** This adapts the terrain colour and applies it over the top of the diffuse color on the object (this is good for grass).
- ▶ **Bending:** This enables bending of the object (if the object permits it).
- ▶ **Hideable:** This controls how the AI will use this object to hide behind it.
- ▶ **PlayerHideable:** If the player is within the bounding box of the object, this determines if the player is hidden from the AI's sight when in the defined stance.
- ▶ **GrowOn Voxels/Brushes:** If true, the painted objects will attempt to align themselves to Voxels/Brushes.
- ▶ **Pickable:** Flags if the vegetation object is pickable.
- ▶ **Brightness:** This applies an additional brightness filter on the object.
- ▶ **Density:** Spacing between the objects when painted (1 = 1meter between objects).
- ▶ **ElevationMin/Max:** This is the minimum and maximum elevation (in meters), which this object can be painted on.
- ▶ **SlopeMin/Max:** This is the minimum and maximum slope angle on the terrain the objects can be painted on.
- ▶ **CastShadow:** This determines whether the object casts a shadow or not.

- ▶ **RecvShadow:** Does the object receive shadows from other objects? (Good to turn off for performance reasons.)
- ▶ **SpriteDistRatio:** This is the ratio in which the object will switch over from rendering the object to the defined sprite.
- ▶ **LodDistRatio:** This is the ratio when the object will switch to rendering a lower LOD.
- ▶ **MaxViewDistRatio:** Depending on the size of the bounding box, this determines the maximum ratio where the player will be able to see the object at all.
- ▶ **Material:** Any custom-defined material for the object.
- ▶ **UseSprites:** If the object should render the Sprites at all.
- ▶ **MinSpec:** The minimum system specification this object should render on.
- ▶ **Layer_Frozen/Wet:** This applies custom Frozen/Wet shaders on top of the object.
- ▶ **Use On Terrain Layers:** This automatically populates the terrain layer with the object with the above defined parameters.

See also

- ▶ The *Terrain Texture setup* recipe
- ▶ The *Placing objects in the world* recipe

Breaking up tiling with Decals

Up to this point, you might have noticed that the terrain texturing can look a bit poor with the amount of tiling that may be seen. In this recipe, we will explore how you can break up some of the terrain tiling with Decals.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Complete the *Terrain Texture setup* recipe

How to do it...

1. From the RollupBar, open the **Objects** tab.
2. Click the **Misc** button.
3. Select **Decal**.
4. Align the **Decal** to your terrain and place it down in an area with a lot of tiling.

5. From the **Decal** properties, click **MTL** (or bring up the material editor).
6. Find **Materials | decals | ground_crack_a**.
7. Apply the material to the **Decal**.



How it works...

Decals are simple texture planes that can be projected against both terrain and objects (such as the Road tool).

There's more...

Here is some additional information about the Decals and what parameters can be set.

Decal parameters

- ▶ **ProjectionType:**
 - ❑ 0 = Planar
 - ❑ 1 = Projects on static objects (Brushes)
 - ❑ 2 = Projects on Terrain
 - ❑ 3 = Projects on both static objects and Terrain

- ▶ **Deferred:** Uses screen space calculation instead (better on performance, but may not look as good)
- ▶ **ViewDistRatio:** Distance Ratio in which the Decal is seen based on the size of the decal
- ▶ **SortPriority:** The priority of which projected texture should be displayed on the top (roads match this priority range)

Decals can have their unique position, rotation, and scale

Decals can have their own position, rotation, and scale, just like brushes.

See also

- ▶ The *Road tool* recipe

Making caves with Voxels

In this recipe, we will explore the basic uses of the Voxel object and how you can create caves in the cliff sides of your terrain.

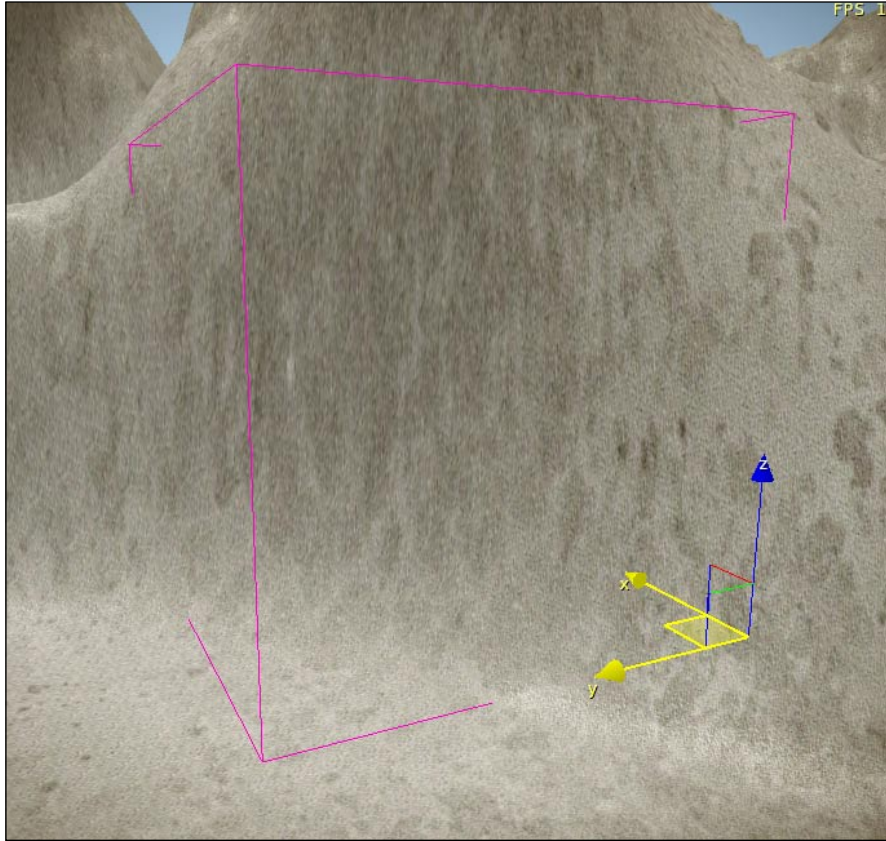
Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Adjust a section of your **Heightmap**, so that you have a mountain side to make a cave from
- ▶ Complete **Terrain Texture Setup** and paint the mountain side

How to do it...

1. From the RollupBar, open the **Objects** tab.
2. Click the **Misc** button.
3. Select **VoxelObject**.

4. Place the **VoxelObject** a little towards the side of the mountain, as seen in the following screenshot:



DO NOT rotate or scale the Voxel object.

5. Select the Voxel object and click **Copy Terrain into Voxel**.

This has now copied the shape of your terrain into the Voxel. You may not see the differences yet, but we will now cut a hole into the mountain using the Voxel Painter to create a cave:

1. Go to the **Terrain** tab in the RollupBar.
2. Select **Voxel Painter**.
3. Select **Soft Subtract**.
4. Set your **Brush Radius** to **15**.
5. Click once around the centre of the Voxel surface.

Again, you may not see anything different, but that is because there are still two layers rendering (Terrain and Voxel). We will now cut holes into the Terrain, so we can see the Voxel cave that is forming:

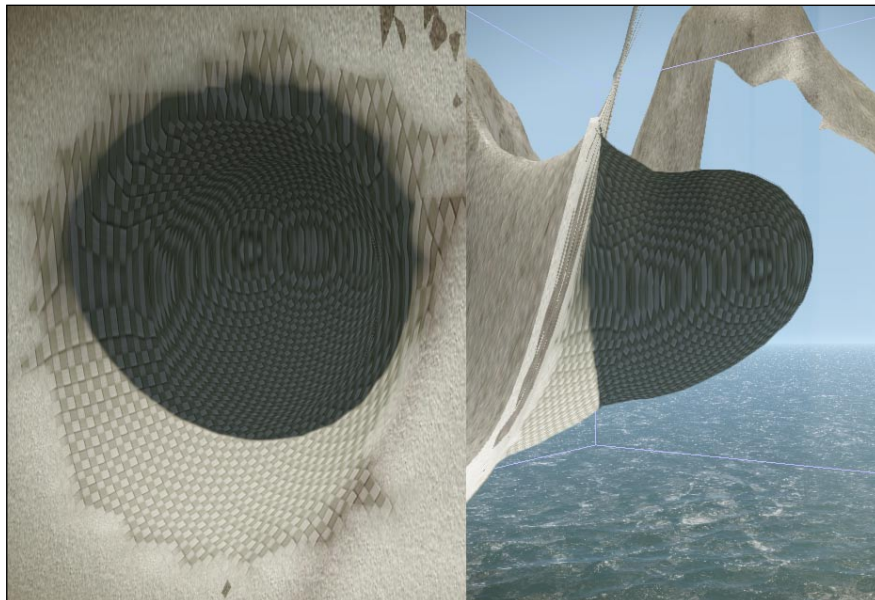
1. Go to the **Terrain** tab in the RollupBar.
2. Select **Holes**.
3. Select **Make Hole**.
4. Increase your **Brush Size** to a reasonable size (**1/4**).
5. Make a hole around the same area as your Voxel (be sure not to cut holes outside of the Voxel object).



If the Terrain tiles are too large for the holes that are created, increase the tile resolution from the Layer Painter.

6. Go back to the Voxel Painter and continue subtracting the Voxel.

A cave like the following should begin to appear:



How it works...

The Voxel tool is a very powerful tool that allows the designers to create caves or even some extrusions from the terrain surface without requiring an artist to create new assets.

There's more...

Enhance the Voxel using the following options.

Soft Create

Soft Create extrudes the triangles of the Voxel instead of bevelling them inside.

Material

This allows the designer to paint the materials from the *Terrain Layer Painter* into the Voxel.

Copy Terrain

Painting with this brush restores the Voxel to the Terrain's original position.

Creating Prefabs to store in external libraries

In this recipe, we will show you how to create a Prefab and look at some of the global uses for them.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Create **the house object again from the** *Editing and Merging Solids* recipe
- ▶ Then place down the **Objects/natural/trees/hill_tree/hill_tree_large_bright_green.cgf** as a brush near **the house**

How to do it...

The house and the tree are just two objects that will be used to demonstrate creating a Prefab.

Before we begin, we must create a new library to store these objects:

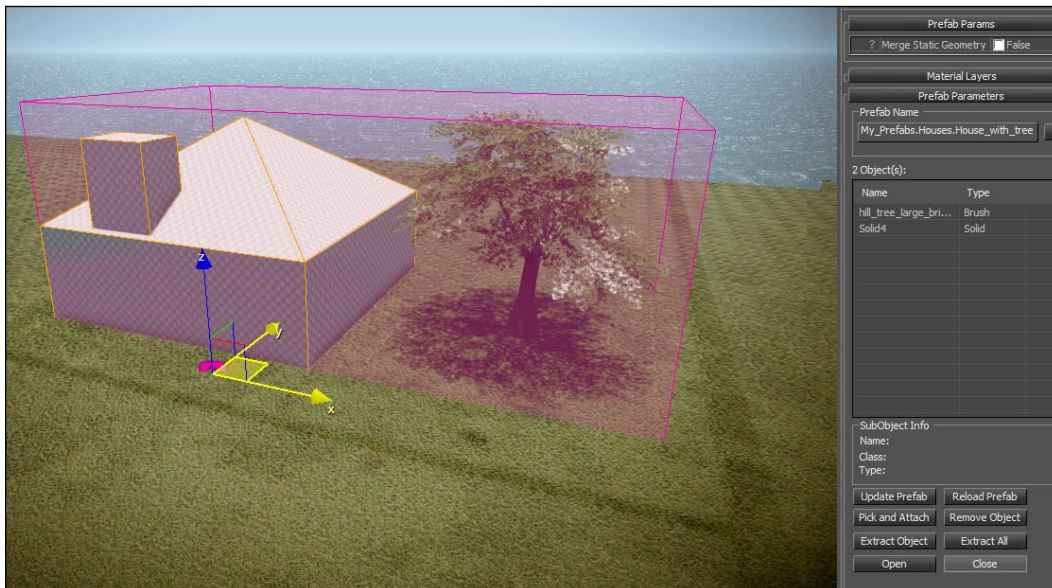
1. Open the **DataBase View** (**View | Open View Pane | DataBase View**).
2. Open the **Prefabs Library** tab.
3. Create a new Prefab library named **My_Prefabs**.
4. Select both the **House** and the **Tree**.
5. In the main toolbar, click on **Prefabs | Make from Selection**.

6. Group = Houses.
7. Name = House_with_tree.
8. Click on the **Save Modified Libraries** in the **DataBase View**.



If you do not perform step 8 and close the level, you risk losing your Prefab!!

This has now created a new Prefab for you:



How it works...

Prefabs are much like groups of any object/entity within the engine. The difference is that they are self-contained groups that can be propagated to all other maps if the Prefab itself changes.

The major benefit of Prefabs is that any logic, sound, art, AI navigation, or additional work may be done on this *grouped* asset without the level designer needing to place a whole new asset every time. Each time the Prefab gets updated, the level designer will see the change once the level is re-opened with the new Prefab library.



Modified Prefabs will NOT be seen in the game mode unless the level has been exported with the latest Prefab in the Editor.

There's more...

Here are some extra functions that can be used with Prefabs.

Extract Object and Extract All

After the Prefab is selected in the level, you can then select an object from the Prefab list and choose to extract that object from the Prefab. This allows the designers to take objects out of the Prefabs – if required – and modify them on their map individually.

Extract All will allow the designers to unpack everything from the Prefab completely. This is normally done to allow designers access to link logic to objects within the Prefab (Prefabs do not allow any links to pass outside of their packed content).

Open/Close

Without extracting the objects, it's still possible to change any of the properties of an object within that group. Simply **open the group and select the object you wish to manipulate inside**.

To close the Prefab, select the group and click on the **Close** button.

Pick and Attach/Remove Object and Update Prefab

You may also choose to pick and attach other objects/entities and put them into the already existing Prefab. Simply align the object/entity to the relative location you wish to have it within the Prefab. Then click **Pick and Attach** and select that object.

Alternatively, you can select an object from the Prefab's list and click **Remove Object**. This will remove the object from the Prefab and leave it where it was located on the map.

Both of these methods affect what is stored in the Prefab, however the Prefab itself is not automatically updated. After you've chosen to modify the Prefab in any way, (Pick and Attach/Remove Object/Modified an Object inside the Prefab) you must then click on **Update Prefab**. This will make sure the Prefab is re-written into the **DataBase View**. To make sure the Prefab library is saved, go to the **DataBase View** and save the **Prefab Library**.

See also

- *The Grouping the objects recipe*

4

Environment Creation

In this chapter, we will cover:

- ▶ Creating your first *time of day* with the basic parameters
- ▶ Adjusting the terrain lighting
- ▶ Using the real-time Global Illumination
- ▶ Adjusting the HDR lighting and the effects for flares
- ▶ Creating a global volumetric fog
- ▶ Creating a night scene with the *time of day* parameters
- ▶ Color grading your level
- ▶ Creating a realistic ocean
- ▶ Improving your sky with the clouds
- ▶ Making it rain in your level

Introduction

In this chapter, we will cover some of the environmental improvements and additions you can make within the CryENGINE 3 Sandbox with levels.

A good deal of the tools covered here will be atmospheric in nature; however, when used in combination with game play elements, this can make for quite a unique and immersive gaming experience for your player.

Creating your first time of day using the basic parameters

In this recipe, we will go through the creation of a basic *time of day* for your level.

A common sunny daylight environment can be easily created using a bright yellowish sun color, a bright blue color sky to be used for ambient lighting, and finally, an angled sun direction to get some interesting shadows.

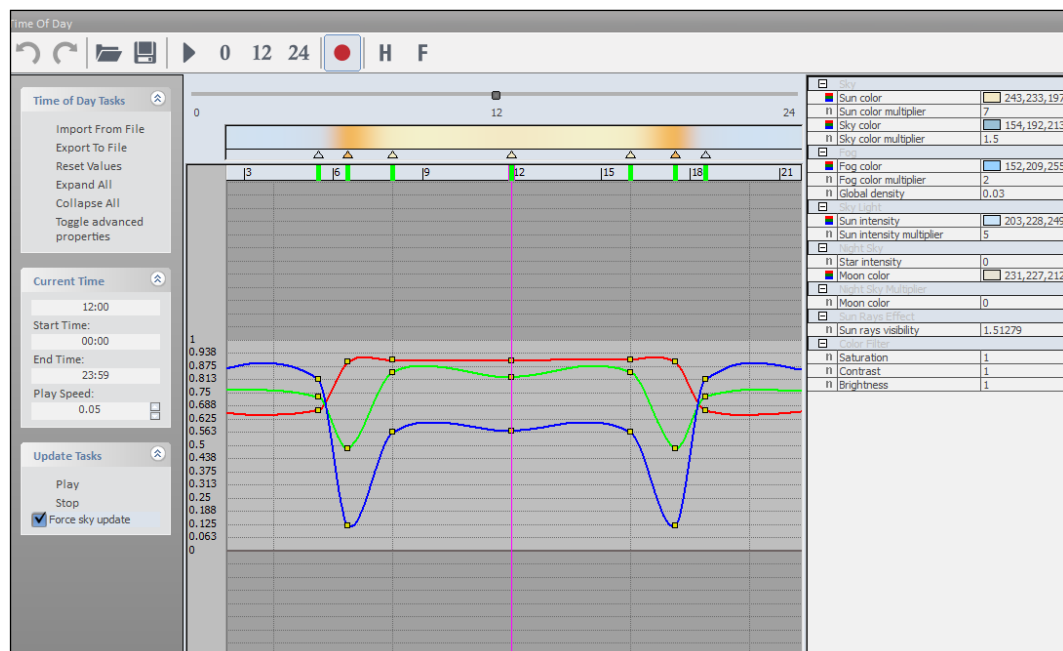
Getting ready

For this tutorial, you should have the forest level `forest.cry` opened. This level is installed automatically as a sample when you install the SDK.

Usually, when creating your own *time of day*, you should use photo reference or fairly specific concept art. Experimenting will allow you to find an interesting mix of values, but in the end you may want to achieve a certain look. So having a goal before beginning to adjust the *time of day* is recommended.

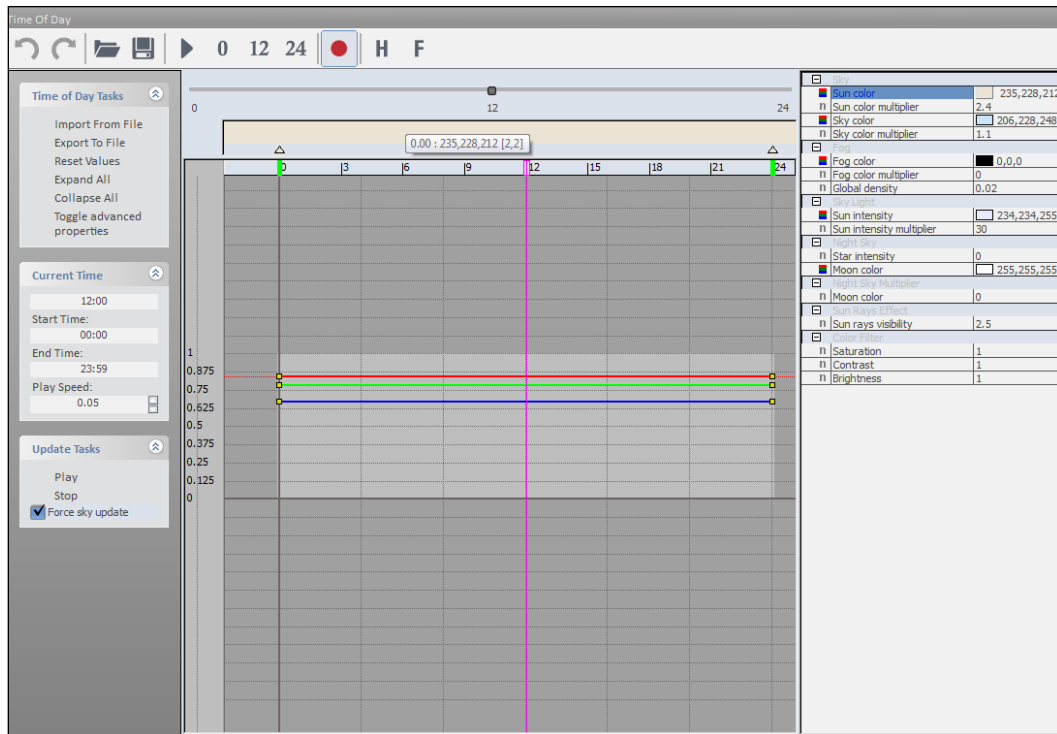
How to do it...

1. Open the **Time Of Day** dialog from **View | Open View Pane | Time Of Day**.



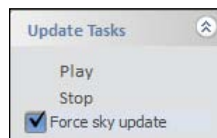
Notice, when you highlight any of the basic parameters in the **Time Of Day** dialog there are already *key frames* created on the timeline. This timeline is represented by the 24 hour slider at the top of the interface. Since the best way to learn to manipulate the *time of day* editor is to create your own, let's go ahead and reset all the values.

- To do this, click **Reset Values** in the **Time of Day Tasks** window.



This will effectively remove all the key frames outside of the default values at **0000** and **2400**.

- Before we adjust the parameters to suit our needs, set **Force sky update** to **True** and ensure the record button is highlighted.



Next, let's set the current time in the level so that we can see the sun.

Environment Creation

4. To do this, click-and-drag the arrow on the slider to late afternoon. If you have difficulty getting the current time accurate, you can type in the current time.
5. Set the current time to **1600**.

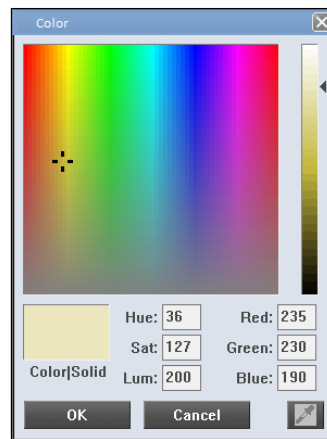


Having the sun intersect some geometry will allow sunrays to be shown more clearly.

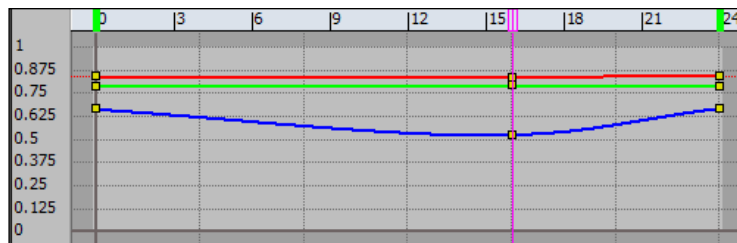



The first setting we will adjust will be the color of the sun. The sun in CryENGINE 3 is an advanced dynamic light. Adjusting the sun color in simple terms will adjust the diffuse color for this light.

6. To adjust the overall color, first click the color **sampler box** and **click-and-drag the black target** to your preferred value. In our case, let's take a realistic approach to lighting this level and set the sun color to a warm yellow tone of **RGB 235, 230, 190**.




7. After clicking **OK**, you will see that new key frames have now been created on the timeline.



 You can see the interpolation of the values from **0000** to **2400**. This will be used in case you have an animated time of day.

8. The next parameter we adjust will be the multiplier for the sun color we previously set. Set the **Sun color multiplier** to a level of **8**.

Next, we will set the sky color.

 Though the setting is called **sky color**, it is more accurately the ambient lighting color for the entire level.

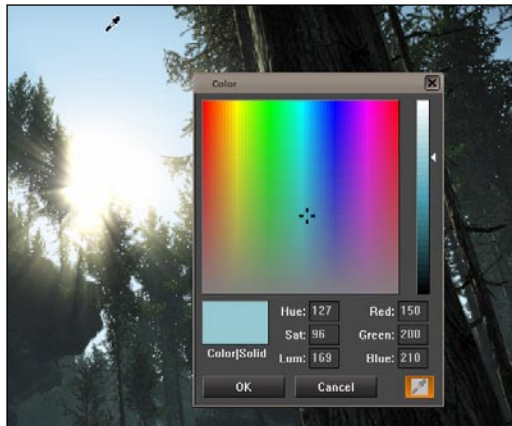
To observe the effects of changing this color, simply look anywhere there is no sun affection on objects from the sun lighting.

Let's first set the color to the currently visible color of the sky:

1. To do this, click the color sampler box for the sky color parameter and click the sample tool.



2. Next, click somewhere on the sky in the perspective viewport, this will sample its color.
3. For this tutorial, set your sky color to RGB **150, 200, 210**.



The sky color multiplier works as the ambient color multiplier. Since we are going for a more realistic looking *time of day*, set this down to **0.8**.

Next is the *Fog* subsection of *time of day*. In our case, we only need to adjust the global density value.



When using a dynamic sky, some haze is already calculated by the sky model. The fog specified by the fog color gets added to that haze. In many cases, the haze may be enough to get properly colored fog for a given time.

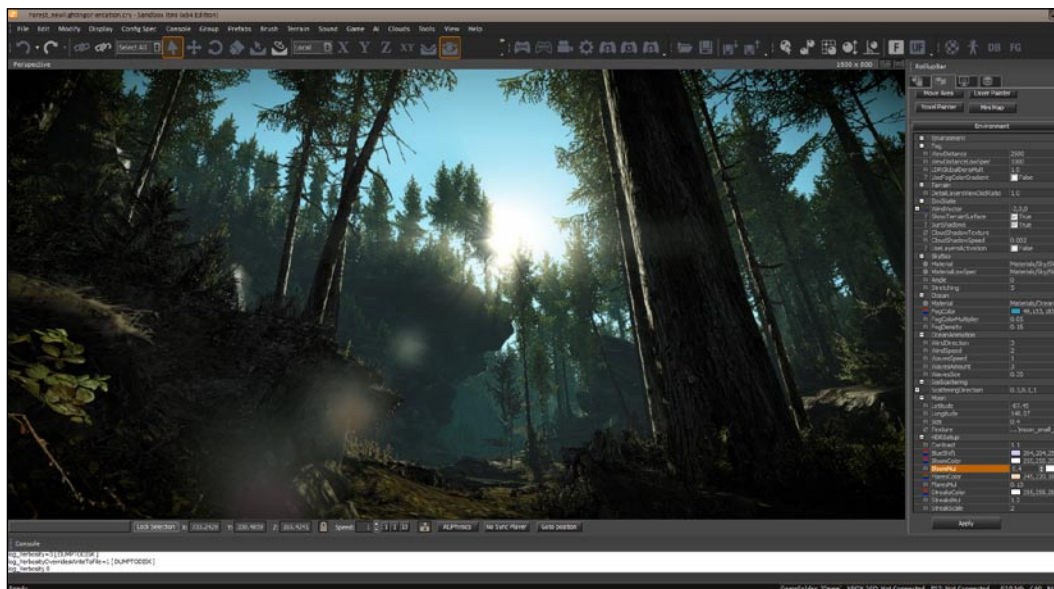
As we do not require a high amount of fog for a pleasant realistic scene, let's set the global density to a value of **0.02**. **The next setting we adjust will be the Sky Light parameters. We** can use a handy feature of the **Time Of Day** dialog to copy and paste the RGB values from the sky color.

Use the sun color for a source color to warm the map or use the sky color to cool the map:

1. Highlight the values in the sky color then right-click and select copy.
2. Next, highlight the sun intensity values then right-click again and select paste.
3. You can also type in the RGB value used in the sky color or sun color. In our case, we will use **RGB 150, 200, 210**.
4. Adjust the **Sun intensity multiplier** to a bit higher than normal to get a volumetric look to the fog, use a multiplier of 50.

For the time being, we will not be setting any parameters for the night sky and night sky multiplier as we will discuss that later. The final setting we will adjust for our basic *time of day* will be the *Sun rays Effect*.

5. This value controls the visibility of sun rays. Higher values will cause brighter rays to appear around the sun, lower values are a bit less stylized but more akin to a realistic look. Set this to **2.0**.
6. The last three parameters shown in the **Time Of Day** dialog are the color filter settings. However, for this process, it is much better to use color grading, which will be discussed later in this book.
7. Leave the color filter settings at default and you should have created a setting that looks like the following screenshot:



How it works...

The sun in the CryENGINE is approximated by a colored directional light without distance attenuation. This is important to understand as its properties are similar to that of a regular light and properties such as color and specular level can be adjusted. However, there is an important distinction to the sun and other lights in the CryENGINE, as the sun uses a technology called cascaded shadow maps.

Sun shadows are achieved in the renderer by splitting up the view frustum into multiple parts that the CryENGINE can handle separately, also known as **Cascaded Shadow Maps**. Additionally, **Variance Shadow Maps** are also used for the lower resolution shadows from the terrain.

Sky light parameters **are solely used to compute the atmospheric appearance of a dynamic sky**. They do not directly affect the rendering of objects in the world (for example, lighting colors and intensities). However, they are used by the engine to compute an approximate fog color to automatically fog the entire scene in the appropriate colors.



For static skyboxes, only the fog color in the fog parameter's group is used.

The **Time Of Day** dialog has a huge variety of settings to simulate realistic and surrealistic lighting effects. Having completed setting up the basic parameters will make it substantially easier to adjust the advanced effects that the *time of day* allows you to adjust to achieve a photorealistic-looking outdoor lighting.

There's more...

You may want to know what the reason for forcing sky update is, or what the record and play functions do.

Forcing sky update to true

Setting the **Force sky update to True** in the **Time Of Day** dialog forces a complete update of the sky light calculations in each frame shown in the perspective view. Should this not be set to true, the sky light calculations will be distributed over several frames and the time it will take to update will be highly dependent on the platform that is running the engine. You should be aware of this when editing some of the parameters within the *time of day* as some will take extra time to update.

Record icon

The **record icon** is highlighted by default. The record button allows the changes made in the parameter's panel to be stored into the levels of *time of day*. This can be used to preview single parameter changes at a time without the worry of writing its changes to the *time of day*.



Play icon

The play icon starts or resumes the playback of the *time of day* sequence in the Editor; if the current time is not within the specified time range (between the start and end time), frame playback begins at the specified start time.



See also

- ▶ The *Adjusting the terrain lighting* recipe
- ▶ The *Using the real-time global illumination* recipe
- ▶ The *Creating a night scene with the Time of Day Editor* recipe

Adjusting the terrain lighting

Having learned how to set up a basic *time of day* parameter, the next step will be to adjust the level's lighting direction and dynamic characteristics using the **Terrain Lighting** dialog. This tool allows you to set the direction from which the sun rises as well as the distance your location is from the North Pole to the South Pole.

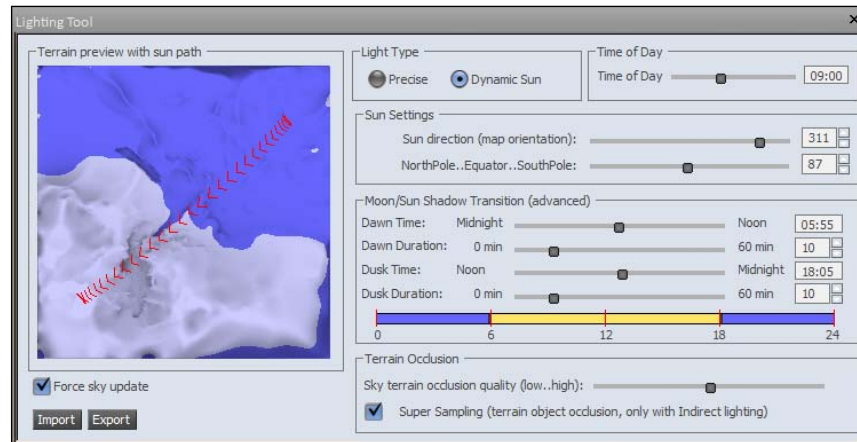
Getting ready

For this tutorial, you should have the forest level `forest.cry` opened. This level was installed as a sample level upon the installation of the CryENGINE SDK.

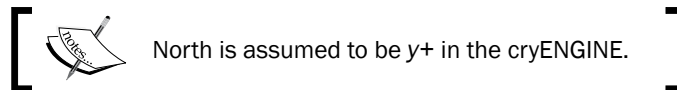
How to do it...

We will now explore the terrain lighting dialog:

1. Open the terrain **Lighting Tool** by clicking the **Terrain** section of the main toolbar and then lighting.



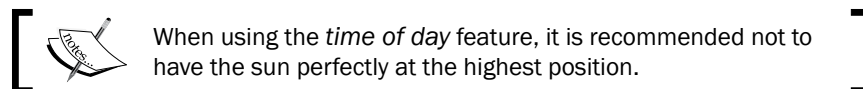
2. Under the **Sun Settings** section there are two sliders. Adjust the first slider called **Sun direction (map orientation)** to the centre most point on the slider. This aligns the sun so that it rises in the east and sets in the west.



3. Next, adjust the second slider named **NorthPole..Equator..SouthPole** to its centre most position.

This will change the *tilt* of the sun; when set at the centre, setting the current time to **1200** will cause the sun to be directly above you. Adding a tilt in either direction will infer that your level's location in reality is closer to either pole of the planet.

4. Adjust the tilt slightly towards the North Pole as this particular environment looks to be in the northern hemisphere.
5. Click **OK** to save your modifications.



How it works...

The *terrain lighting* works in unison with the *time of day* settings to give the level its outdoor lighting characteristics.

It is good to have some rotation on the sun so that the shadows are not aligned with the world direction and thus, the objects that may be aligned. This is all to make man made buildings and other level objects look better as they are often world aligned and built straight up.

There's more...

You may want to know more about the transitioning in shading from the sun to the moon or how to adjust the terrain ambient occlusion.

Terrain ambient occlusion

Adjust the quality of the terrain occlusion by adjusting the slider at the bottom of the **Terrain Occlusion** dialog box. Increasing the **Sky terrain occlusion quality (low..high)** will adjust the amount of light occlusion and render a real-time preview. Terrain occlusion is often used to create the effect of indirect lighting.

Transition shading from the sun to the moon

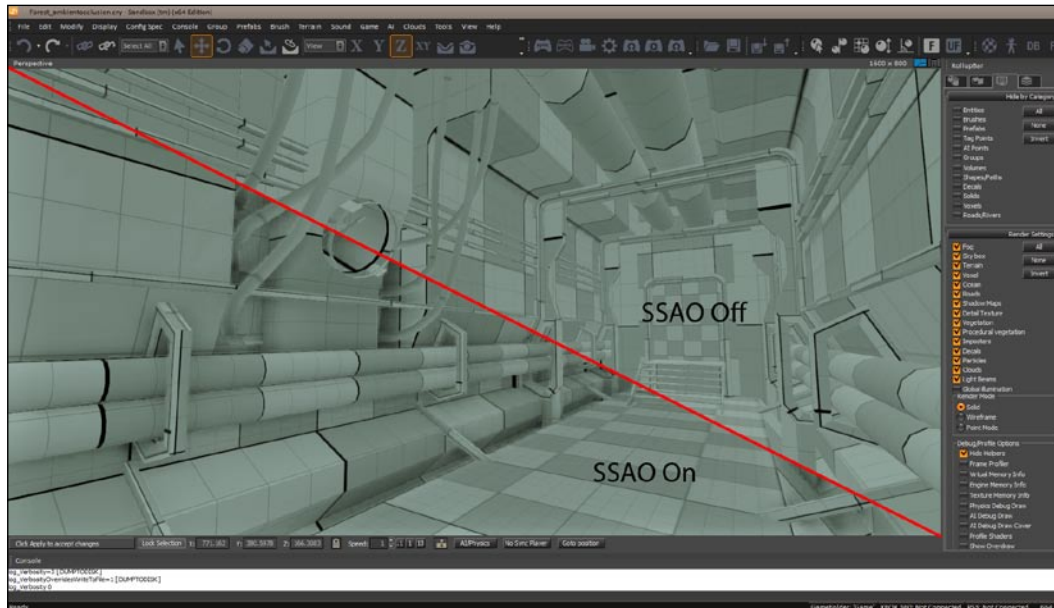
A fundamental principal of the *time of day* is that there can be either sun or moon lighting, but not both. You need to adjust the **Moon/Sun Shadow Transition (advanced)** parameters to adjust the time when the sun should be active and when the moon lighting should become active and vice versa.

- ▶ **Dawn Time:** This sets the time when the sun should rise
- ▶ **Dawn Duration:** This changes the duration of the moon to sun lighting transition phase
- ▶ **Dusk Time:** This sets the time when the sun should set
- ▶ **Dusk Duration:** This changes the duration of the sun to moon lighting transition phase

SSAO (screen-space-ambient occlusion)

Ambient occlusion is a well-known technique in the movie industry to approximate the effect of indirect lighting. Usually, it needs to be computed during the game production time and it only supports static scenes.

CryENGINE 3 uses a technique that is very suitable for real-time and can support dynamic scenes. Because it does not have to be pre-computed, there is no production time lost in creating this effect.



With **SSAO (screen-space-ambient occlusion)**, you can expose the geometric detail of scenes and ensure that its complexity is retained. This is a screenspace effect, so performance is not affected by scene complexity.

You can adjust the SSAO amount multiplier in the **Time Of Day** dialog.

See also

- ▶ The *Generating a procedural terrain* recipe in *Chapter 2, Sandbox Basics*
- ▶ The *Editing HDR lighting and effects for Flares* recipe

Using the real-time Global Illumination

Global Illumination (GI) is an integral part of most of the current and next generation games and will undoubtedly be a major point of interest.

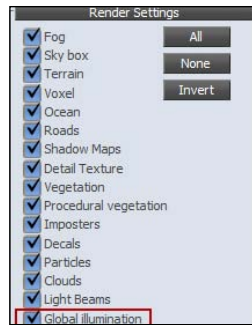
GI is at its core a lighting technique that is meant to enrich the picture and add to the overall image quality, and thus, its perception by the user.

In this section, we will learn how to use the Global Illumination as it suits the purposes of immersive and dynamic gaming quite well due to its real-time nature.

Getting ready

You should have the forest level `Forest.cry` open in the Sandbox Editor.

Under the **Options** panel in the **RollupBar**, you must have set **Global illumination** to true/enabled.



How to do it...

1. Open the **Time Of Day** dialog.
2. Now that we are moving into the advanced environmental settings, click the **Toggle Advanced** properties. **This will now expose all the properties available to the Time Of Day** dialog.

The only attribute we are concerned with, in this example, is the **Global illumination multiplier**.

3. To observe the affected areas of Global Illumination, set the multiplier to its maximum of **100**.



4. To finalize the settings for our level, set the **Global illumination multiplier** back down to a value of **2**.

How it works...

The majority of current games and game engines use various approaches to some kind of previously baked or pre-computed global illumination. **Using pre-computed global illumination** has many restrictions **and imposes some inherent limitations**. **One important limitation is that**, in most outdoor situations, you would be forced to maintain static lighting conditions and/or static objects; otherwise the pre-computed texture would not fit the scene.

For all intents and purposes, the CryENGINE 3 can be mostly considered a baking-free game engine. A major unique point of the CryENGINE 3 is that it provides physically-based phenomenon such as dynamic time of day and object breakability to enrich gaming experience and also to simplify game production process, which means that using many of the pre-computational approaches would not compliment the engine.

There's more...

You will definitely want to experiment with some of the more advanced console variables that adjust the application of GI to your level.

Advanced GI Cvars

If you plan to enable GI for your levels, it is important that you know the advanced GI settings available from the console and their uses:

- ▶ **e_GIAmount = (X)**: This **will multiply with the Global illumination multiplier** set in the **Time Of Day** dialog. The default is **1**.
- ▶ **e_GIMaxDistance= (X)**: This **will extend the range of the Global illumination effect** from the camera. The default is **50**.



Closer distance is equal to higher quality.

- ▶ **e_GIOffset = (X)**: The **GI is an offset forward of the camera to a certain amount**. This can be adjusted to **0** with a small impact to performance.

Continue to the next recipe to edit the HDR lighting for your level.

See also

- ▶ *The Editing HDR lighting and the effects for Flares recipe*

Editing HDR lighting and the effects for flares

HDR Lighting is an important aspect of level lighting. In this recipe, we will adjust the environment variables pertaining to some of the advanced HDR effects available within the engine.

In this recipe, we will explore the **Environment** settings available in the **RollupBar**.

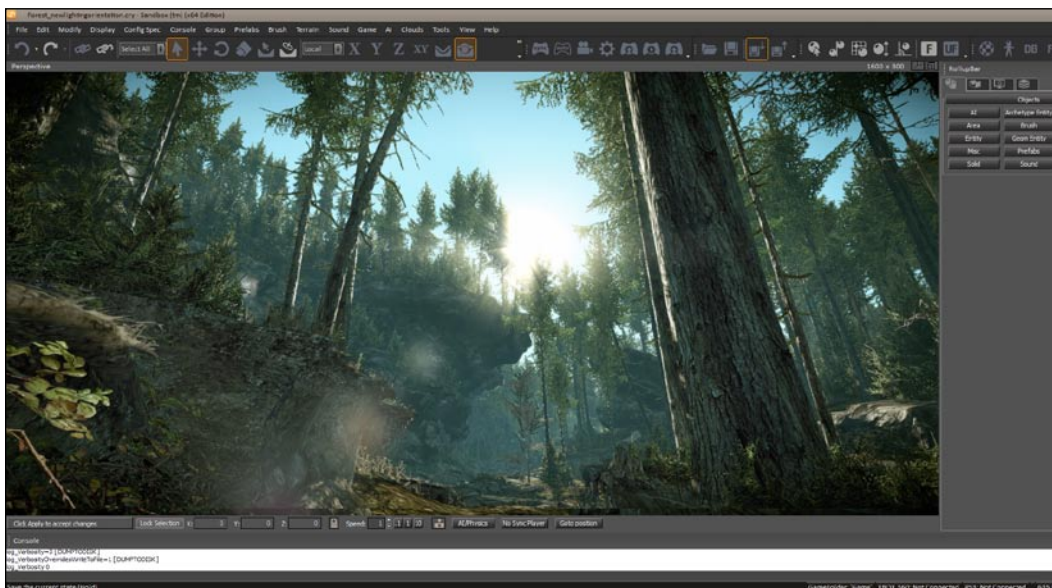
Getting ready

For this recipe, `Forest.cry` should be opened.

How to do it...

As we are currently using an HDR compatible sky in the forest level, we can adjust the HDR in the **Time Of Day** dialog.

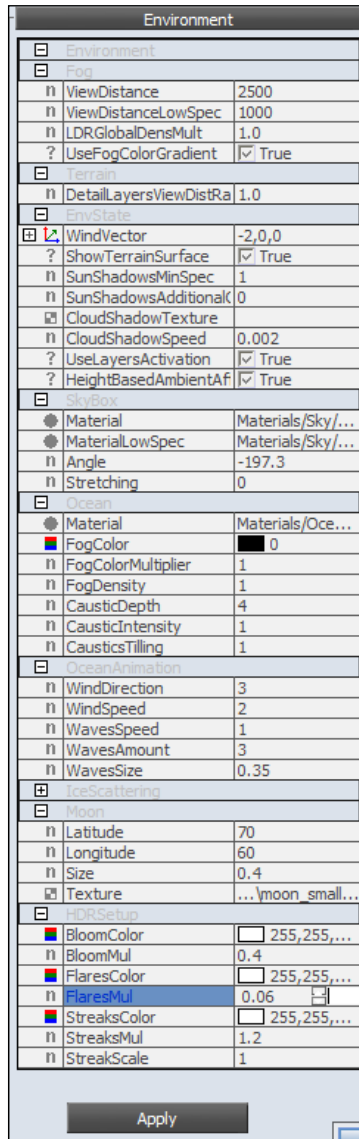
1. Open the *time of day* and the very top parameter is the **HDR Dynamic Power Factor**. **Most of the times, to enhance the HDR lighting during daytime scenes, this should be increased.**
2. Set this to a conservative value of **2** to retain realistic lighting conditions.





The HDR lighting dynamic power factor can be increased at night to enhance flare effects.

- Having adjusted the multiplier, let's now move on to the **Environment** parameters available to us in the **RollupBar**. Open the following from the **RollupBar**:



4. At the bottom-most of this rollout is the **HDRSetup** section. The first thing we will adjust is the **HDR Contrast**. In general, increasing the HDR contrast can heighten the users' perception of intensity. Decreasing it can also be useful in extreme lighting situations. Set it at **1.1** for this example.
5. We will not adjust the color for the **HDR Blue shift** or the **BloomColor** as the default values are quite sufficient for our needs.
6. Adjust the bloom multiplier (**BloomMul**) to **5** and notice the effects. Often heavy bloom is used to induce a dreamy type effect to the player. It increases the visibility of flares.
7. As the flare color is heavily dependent on the view of the player, it is recommended to use a setting that has a fairly bright but neutral color. In our case, let's select a color of RGB **24, 220, 185**.
8. The final property to adjust is the flares multiplier. This controls the overall intensity of the flares. As this is a mid-day scene with a decent amount of HDR lighting, we should keep this quite low. Adjust the flares multiplier (**FlaresMul**) to a more realistic value of **0.07**.

You have now adjusted the HDR settings in the *time of day* and the rollout bar!

How it works...

These settings are saved within the `.cry` file and thus to the game once it is exported to the engine.

Using these settings you can give the player a dynamically lit, realistic or even stylized environment by pushing some values, such as the bloom multiplier, to extreme values.

There's more...

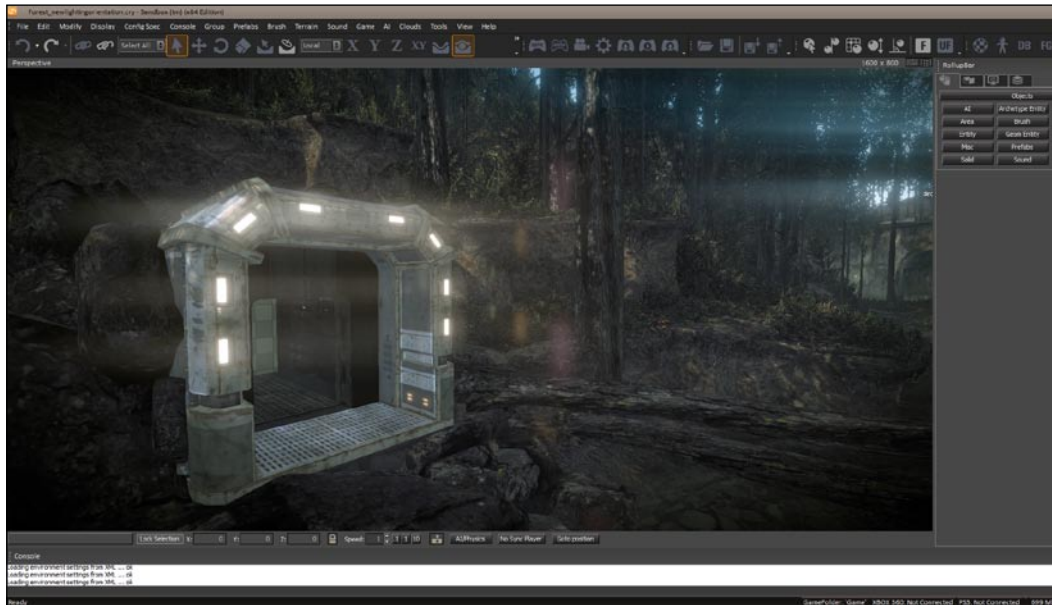
You may want to know how to manually create flares or coronas using lights or what texture effects produce pronounced HDR bloom.

Lights with flare effects

When a light entity is placed in the level, it is able to produce a flare effect. However, it does require a material with the correct **Light.Flare** shader.

Glow texture effect produces bloom and flares

While adjusting the setting for HDR flares be mindful of the asset within the level. Some assets may use a glow effect or texture that can produce strong flares and streaks.



This image shows the extreme values in the HDR settings but communicates well the reason the developer must check different assets within their levels as the lighting settings are adjusted.

See also

- ▶ The *Creating a night scene with time of day parameters* recipe
- ▶ The *Creating a global volumetric fog* recipe

Creating a global volumetric fog

In this recipe, you will interact with the HDR effects and learn to manipulate the global fog and other fog effects such as volumetric fog.

Getting ready

You should have `Forest.cry` open in the Editor and have completed the *Creating your first time of day using the basic parameters* recipe earlier in this chapter.

How to do it...

As we will soon be adjusting the fog color, to be able to visualize it easier being multiplied on top of the haze created by the HDR sky, we must set its multiplier:

1. Open the **Time Of Day** dialog and ensure that the advanced *time of day* settings are displayed.
2. Set the **FogColorMultiplier** to **10**, so that we can easily see our color changing.
3. Next, select a color for the fog.

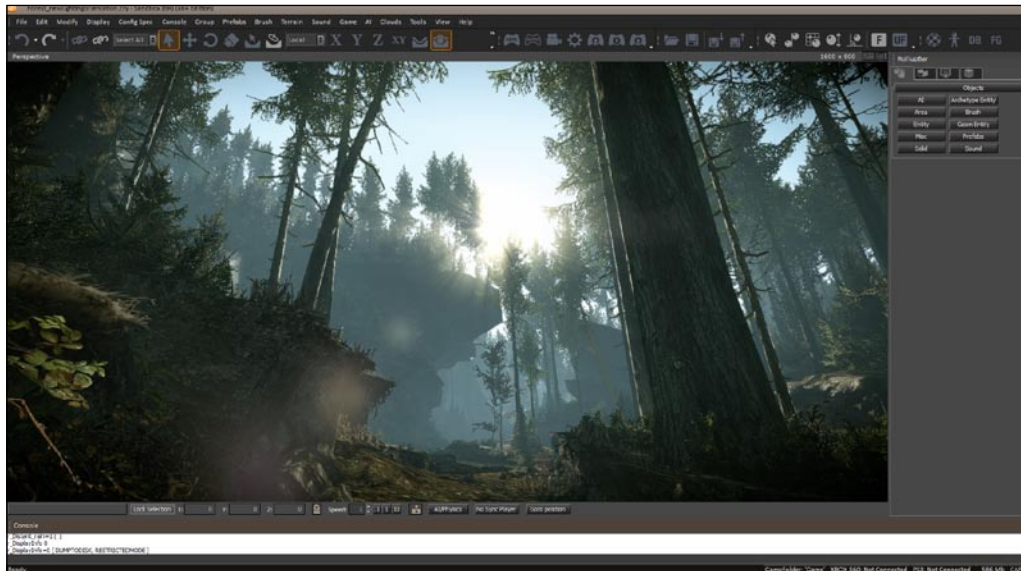


A good technique is to copy the color of the sun to warm the map or the color of the sky to cool it.

4. In the following final screenshot, I have **used the sky color of RGB 150, 200, 210** for the fog.
5. Next, let's adjust the atmosphere height. This parameter defines the atmosphere height **above the sea level**. So, in reality the higher the atmosphere is, the less visible the effect of fog layers will be.
6. Let's set our **atmosphere to 8000**. Since we have raised the atmosphere, the fog in the distance will be too bright and we will need to adjust for this.
7. Set the **FogColorMultiplier** down to a more comfortable value of **5**.



If you still find that the density of the fog is too thick, then adjust the global density.



You can now see that setting good volumetric fog can increase the believability of a level and especially highlight the silhouettes of distant objects.

How it works...

Global volumetric fog in the CryENGINE allows you to model almost all aspects of an atmospheric *Aerial Perspective*. It simulates particles/aerosols distributed uniformly along the ground plane and falling off exponentially from a height (based on height above the sea level). The fog integral between the viewer and each pixel in the scene gets properly solved taking the two mentioned distributions into account. Additionally, it accounts for sunlight scattered into the view ray to produce halos around the sun in foggy environments.

There's more...

You would have noticed other settings under the **Fog** parameters in the **Time Of Day** dialog such as **Density offset**. You might also want to know how to enable and disable fog rendering for ease of editing.

Density offset

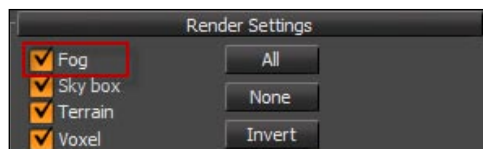
The **Density offset** parameter shifts the fog density per pixel, by the given amount. As a result, objects that are close to the viewer appear less foggy.



Adjusting this effect should be used sparingly as there is a realistic trait to this visual effect.

Enabling or disabling fog in the Render Settings

You can easily disable the rendering of fog in the Editor without adjusting the *time of day* settings. Do this by navigating to the **Display** tab in the **RollupBar** and unchecking **Fog**.



See also

- ▶ Go back to the *Terrain Lighting* recipe earlier in this chapter to see the effects of sun lighting on the fog settings
- ▶ Continue to the next recipe to create a custom skybox for your level

Creating a night scene with time of day parameters

In this recipe, we will be creating a night scene utilizing the advanced parameters of time of day.

Getting ready

You should have `Forest.cry` open in the Editor.

How to do it...

Let's create a night time atmosphere for our level:

1. Open the *time of day* editor.



You can back up your time of day as a `.tod` file with the export time of day tool.

As we will only be worrying about the important values that will pertain to creating a night scene, let's reset the current values.

2. Click **reset values**.
3. Set the **time of day to 21:00**, which **we can safely call night**.
You will notice that the sun goes below the horizon but we can now adjust the position of the moon texture, which – for all intents and purposes – will be used as the "sun" in our night scene.
4. The first value we should change is the moon color in the **Night Sky Multiplier** section of the time of day dialog.
5. Set the moon **color to 2**. Now **that the moon is visible we must adjust the moon texture's position and size**.
6. Open the **Environment** tab in the **RollupBar** and locate the moon environment properties.

[-] Moon	
n Latitude	-15
n Longitude	35
n Size	0.7
i Texture	...moon_small_glowing.dds
[-] HDRSetup	

7. Adjust the longitude and latitude to change the position of the moon in the sky to wherever you want it, or you may copy the parameters in the image.

- Now that we have our basic setup, we need to **adjust the overall coloring and lighting** of the scene.



Feel free to add variations to these values as not all the available options are used here.

- Set the sky color multiplier to **0.4**, which will dim **the entire maps ambient lighting**.
- Next set the **Zenith** color in the **Night Sky** parameter to a suitable night sky color of **40,54,51 RGB**.
- Now we can add some stars to the scene by increasing the **Star** intensity in the **Night Sky** parameter to its maximum of **3**.



If you have Depth of Field Enabled it can be difficult to see the stars at times.

- As we want no direct lighting, let's set the sun color **multiplier to 0**.
The final touch we will make is to add some fog to the scene to highlight the silhouettes of distance objects.
- Set the fog color to a fairly dark **color of 61,80,84**.
- Now, adjust the Fog Color multiplier to **1**.
- Finally, to retain the silhouettes of objects set the global density to **0.2**.



If you find the fog too bright on your particular monitor or gamma setting, you can adjust the fog color multiplier further down to **0.5**.

You now have a realistic night time of day!

There's more...

There are many other properties available to you when adjusting for a night scene. Some of these are explained as follows.

SSAO contrast and amount

To enhance your night scene you will likely want to increase the SSAO (screen-space-ambient occlusion) amount and its corresponding contrast.

HDRSetup parameters at night

It is likely that you will need to adjust the HDRSetup parameters to less extreme values than during the daytime in your level. HDR effects such as flares and coronas will be much more obvious and visible to your player at night.

Moon and corona color and scale

If you wish to have a more surreal scene, you can adjust the moon color to different extremes of colors; it should be noted that this color is simply multiplied onto the color of the moon texture. The corona scale and color settings will allow you to highlight your moon and its surrounding sky area with more or less lighting.

See also

Having now created a whole atmosphere, go to *Chapter 3, Basic Level Layout* to start laying out your night time level.

Continue to the next recipe to color grade your entire night scene to emphasize certain colors or styles.

Color grading your level

Color correction is a post processing effect that takes a frame rendered by the engine and changes its output of colors in various ways.

This effect is typically used in film to enhance scenes with effects such as Hue and Saturation, Contrast and Brightness, Luminance and Color Curves, and so on.

Getting ready

Take a reference image that hasn't been color corrected yet and shows a wide range of colors from an open level. Avoid using high resolution image captures though, as they won't improve color correction quality, but rather only increase the processing time when saved later. In this example, I captured a 1280x720 TGA image.

Copy/paste **THE COLOR CORRECTION LOOKUP REFERENCE CHART** into this reference image using Photoshop. It is important to use the color chart supplied by Crytek, as the resource compiler will need this file later on, to detect and extract the chart.

Flatten all layers, and save the file before beginning this tutorial.

How to do it...

Let's learn how to color grade our levels.

Begin by making the desired color adjustments in Photoshop. In this example, apply a **bleach bypass** color gradient to my scene:

1. When color correcting the reference image, ensure that the changes are affecting the added **THE COLOR CORRECTION LOOKUP REFERENCE CHART**.
2. Once the desired changes have been made, you may now save the new image as a CryTIFF file in your root directory under Textures\colorcharts\. **It is essential** that you follow the naming convention: filename + _cch.tif.
3. Having now created our modified **COLOR CORRECTION LOOKUP CHART**, let's integrate this into our scene.
4. First, set the **console variable r_ColorGrading** to **1**, which enables it.
5. **Next**, set the **cvar r_ColorGradingCharts** to **2**.
6. Setting **r_colorgradingcharts** to **2** works **similar to setting it to 1**, but in addition it displays debug information in the upper-left corner of the perspective view with the color chart lookup table, blending information and name of any static color charts that are loaded.
7. Finally, in order to load and use a **static color chart that overwrites the** dynamic blending results mentioned earlier, use the console command **r_ColorGradingChartImage** and specify the path to the color corrected reference image.

For Example, `r_ColorGradingChartImage textures/colorcharts/
filename + _cch.dds`.

How it works...

Color correction in CryENGINE 3 takes a reference image that gets color corrected by a beautification artist or a designer. This image includes a default color lookup chart so all transformation steps applied to the image can later be reconstructed into the scene through the color correction post effect.

When activated through the console with the parameter **r_ColorGradingChartImage**, it loads the defined color chart image.

There's more...

You may want to know more about why the **_CCH** naming convention must be used or how to capture frames directly from the engine for color correcting.

_CCH naming convention

If you follow this naming convention, the resource compiler should automatically pick **ColorChart** as the preset to be used. Make sure that the image is not tiled in the Resource Compiler. If you wish, you may generate the output manually in order to create a file with a **.dds** extension.

Capturing TGA images as reference images

As some artists may prefer to use uncompressed TGA images to adjust the settings, they can be output from the editor using the capture frames command:

- ▶ `set capture_file_format to tga`
- ▶ `capture_frames_once to 1`
- ▶ `capture_frames to 1`

This will save a tga screenshot of the perspective viewport to the **CaptureOutput** folder by default. This can be adjusted using the **capture_folder** console command.

Debugging visual glitches

If you encounter visual glitches on the PC, please ensure that you haven't overwritten 3D settings for either the editor or the game launcher in the control panel of your video driver. This can be a major source of visual artifacts.

See also

Before color grading, you should be happy with the overall *time of day* settings. Go to the *Creating your first time of day* recipe earlier in this chapter, if you want to make further adjustments.

Creating a photo realistic ocean

In this tutorial, we will create our own custom ocean material and apply it to a level. As the ocean is commonly used in games as an interactive game play element, we will also cover the underwater fog settings to make underwater game play enjoyable for our player.

Getting ready

The `Live_Create_Small` level should be open in the Editor. Your **Time Of Day** dialog should be open in the editor and set to a time of **0630**. You should also set it to a very high specification to observe the advanced tessellation features.

How to do it...

Let's create a new photo realistic ocean for the level:

1. First, to see the currently applied ocean material, open the **Environment** tab in the **RollupBar**.
2. Under the **Ocean** parameter you will see four parameters.
3. First, let's apply our own **material**.
4. Click on the **browse** button in the **Ocean material parameter**. This will open the currently applied material.
5. Create a new material and name it `my_ocean.mtl` and save it under your root directory `/materials/ocean/`.
6. Next, click **apply** in the **Ocean** material parameter in the **RollupBar**. You will notice that there is no water yet. This is because the **Ocean** requires a specialized shader to render.
7. To set the shader you must edit your newly created material `my_ocean.mtl`, set the shader parameter in this material to **water**.

Now that we have some basic water, we should apply a bump map to create the small ripples and waves on the water's surface. This will allow some refraction to occur, enhancing the overall effect.

1. For now use the texture **located in** `textures/defaults/oceanwaves_ddn.dds`.
2. To finalize the material ensure that the **Sunshine** tickbox is active in the **Shader Generation Parameters**.



Experiment with different shader settings for more stylized and extreme types of water.

3. Now, let's adjust the **Ocean** settings available to us in the **Time of Day** dialog.
4. Then locate the advanced **parameters**. **Adjust the Ocean Fog Color to a pleasant blue** for this particular scene **107, 217, 241** RGB.



The **Ocean Fog Color** is the overall color setting for the ocean; you can set this to dark blue for deeper water and even green for polluted water.

5. It is easy to observe the changes of the fog **color multiplier and fog density and its** overall effect on visibility by moving your perspective view underwater.
6. After you have moved your camera to an appropriate view, adjust the fog color multiplier and fog density to **0.1** and **0.02** respectively. This setting gives a decent color on the surface of the water as well as a good view distance underwater.

Our final changes will be to the level's environmental affection on the ocean material, which we applied earlier.

Under the **Environment** tab in the **RollupBar**, locate the **Ocean Animation** parameters:

1. We **should first adjust the direction of the virtual wind being applied to our water**. To do this, first set the **WindSpeed** to **8** so as to be able to clearly see in which direction the wind is animating the water.
2. Next, set the wind direction **to a desired value, in this case set it to 4**.
3. Bring the wind speed back to a realistic **value around 3**.
4. Finally, we will adjust the size of the waves.
5. Extreme settings will cause malformations in the waves, so observe these changes closely.
6. For our purposes, a slightly higher than normal wave height can be achieved by setting **Waves Size** to **0.5** and by setting the **Waves Amount** to **1**.



If you want to achieve choppy water conditions, then simply increase the waves amount.

How it works...

The environmental parameters of the water and its corresponding *time of day* parameters are saved within the level.

Using latest graphic card shader programmability, CryENGINE does not suffer from Z buffer intersection with the terrain, but rather gets nice beach foam and underwater fog.

There's more...

You may want to know how the *Caustics* rendering works or how to animate the water parameters. You might also want to know about the technology used in the CryENGINE for water free from transformation rendering.

Animating water parameters

The *time of day* parameters for the *Ocean* can be animated just like any other value in the *time of day*.

Caustics

Caustics are used by default in CryENGINE 3. **Caustics are rendered as material layer, so all objects below water can be affected in and changed in real time.**

- ▶ Very important for realistic water look
- ▶ Extra rendering pass, can handle opaque/transparent, can be done in a deferred way, but we lack some data such as per pixel normals to give an interesting look
- ▶ Based on real sun direction projection
- ▶ Procedural composed using three layers
- ▶ Chromatic dispersion used
- ▶ Darken slightly to simulate a wet surface
- ▶ Stencil pre-pass helps a little bit on performance

Free form transformation FFT water

CryENGINE 3 animates the water through an FFT algorithm on the CPU. There is a unique advantage to this as we get even physics interaction, entities, or objects floating on the waves reacting when you jump on them and this works very naturally.

- ▶ Used statistical Tessendorf animation model
- ▶ Good tillable results for variety of usage: calm, choppy, and so on
- ▶ Computed on CPU for a 64x64 grid, Pirates of the Caribbean 3 used 2k x 2k
- ▶ Upload results into a texture volume
- ▶ Vertex displacement on GPU



Lower HW specs uses simple sin waves sum.

Additionally, normals maps translation – four normal maps layers

- ▶ High frequency/low frequency
- ▶ Animated along water flow direction

Improving your sky with clouds

Clouds are just one example of the various object types and features supported in CryENGINE 3.

Getting ready

You should have the level `live_create_small` open.

How to do it...

Let's use the different cloud entities available to us to populate our sky with:

1. Drag a **Cloud** entity into the level.
2. Place it at above **1000** meters (**Z** value = **1000**).
3. Make sure that the **Cloud** entity is selected, so that its properties are displayed in the **RollupBar**, then click the **MTL <No Custom Material>** icon.
4. This opens the **Material Editor**, where you can select the the desired **Cloud Material**. Make sure that the **Common.Cloud Shader** is selected in the **Material** settings.
5. Scale it by selecting the **Scale** tool from the **Toolbar**, making sure to select all three axes.
6. In order to enable cloud movement, set the **AutoMove** property to **True** and set a fade distance to **100** to make them fade in when they reach the edge of the box in which they are moving.
7. Then, you need to define how big the box (**SpaceLoopBox**) – in which they are moving from one end to the other – should be. **2000** is the default value for all three axes.
8. Finally, you need to set the speed at which they will move. For normal clouds, you can use **5**, and for storm clouds, about **15**. Experiment a bit, but remember that if you want to select a cloud, it's wise to turn off **AutoMove**; otherwise, it can be difficult to select the cloud.

The Distance Cloud Tool allows you to place planes in the sky that look like clouds if the right texture is assigned. They are great for creating a realistic distant sky setup for your level without causing huge render slowdown issues. Distance Clouds are basically horizontal planes placed into the sky. Unlike other types of clouds they cannot be flown through and thus, they're mostly suitable for filling the sky in at far distances (horizon) or high altitudes. However, this limitation also allows for more advanced shading at a cheap price (simple single pass shader, no impostors needed).

1. Drag the **DistanceCloud** entity to the level.
2. Scale it with the *Scale* tool to about **100**, then assign one of the distance cloud materials (all materials with distance clouds shader in materials/clouds folder can be used). Distance clouds are 2D, so they should be placed far away or above so that the player does not see that they are flat.
3. Finally, to place a cloud with nice shading, drag a **Volume entity to the level**.

4. With the **VolumeObjectFile** set to **Libs/Clouds/Default.xml**, select a cloud material in the **Material Editor**. Make sure that the **Shader** is set to **VolumeObject**. The cloud can now be scaled as needed.

The difference between the normal 3D cloud and a **VolumeObject** is that volume object clouds have correct self shadowing. Do not place too many volume object clouds because they greatly affect performance.



If you assign a new material, ensure that it has the correct shader applied. For volume clouds, you have to assign the **VolumeObject** shader.

How it works...

There are four distinct ways to visualize clouds in the engine. This is because there is often the requirement for a mix of types to achieve certain effects or some types might not have to be used. The four ways of achieving clouds in CryENGINE are:

- ▶ Painting clouds into a skybox, which is very controllable for the artists. The method is good for slower graphic cards as it only requires texturing, but it only allows static images and it will not react when changing the *time of day*.
- ▶ The *imposter* clouds are rendered in 3D to intermediate textures and requires update after every few frames. They are slower and consume more or less memory at runtime depending on the view.
- ▶ The *distance* clouds which render faster, have constant memory requirements, support *time of day* shading, but are not real 3D. **A viewer that went near to them would see that they are a flat plane.**
- ▶ The *volume* clouds are true 3D; they even intersect with the geometry correctly and have a constant memory requirement. The downside to rendering them is that they are slower than the rest and in comparison look a bit blurry.

There's more...

You may want to have cloud shadows simulated in your level, which you can find out how to do next.

Cloud shadows

Clouds don't cast real-time shadows, but there can be a (moveable) texture used that is casted on the entire level, creating the illusion that clouds cast shadows.

Use the default cloud shadow texture (**textures | clouds | cloud_pattern.dds**) and adjust it in Photoshop by painting it white in the areas where you don't want to have shadows.

Set the **Cloud Shadow Speed** to **0.001** for fast moving clouds and **0.0005** for normal clouds.

See also

- ▶ Continue to the next chapter, *Chapter 5, Basic Artificial Intelligence* to make it rain in your level, which will compliment the placement of clouds.

Making it rain in your level

Rain can be an essential part of your level. There are some advanced rain techniques available to you within the CryENGINE that will allow you to enable object interaction with the rain and localized wet areas. This recipe will change a map to a rainy scene.

There are two distinct techniques to achieve rain in the CryENGINE. The following example will explore both the particle-only technique and the deferred rain entity.

Getting ready

You should have the forest level `Forest.cry` open in the Editor.

How to do it...

We will begin by using the particle-only technique to create our first rain:

1. Drag-and-drop the **ParticleEffect** entity into your level from the **Particle** tab of the **DataBase View**. The particles are named as follows:

```
rain.rain.space_loop = (libName).(subfolderName).(effectName)
```
2. The settings for how the **rain.rain.space_loop** functions, can be adjusted via its property settings on the **Particle** tab of the **DataBase View**.
3. Adjust the **Countscale** of the **Particle** effect to **4** (be aware that this value multiplies with the count value of the particle).
4. Next, set the size to **0.5** for a smaller and more realistic rain drop.
5. Next, let's limit this effect to a certain area **of our level**.
6. First, make a shape and use the pick tool of the shape in the **RollupBar** to select the spaceloop particle system we just created.
7. This will cause the particle effect attribute *active* to be activated and deactivated if the player crosses the border of the shape.



You may have to set **active** to **false** to have the particle disabled when you enter the game initially.

The next effect we will use is the **PostEffect Rain** that is new to CryENGINE 3. This is a rain effect entity that will apply a post effect within its radius. These entities are placed just like normal entities:

1. Remove the particle effect we created earlier to disable the particle only rain.
2. Drag the **Rain** entity from **Entity | Environment** in the **RollupBar**.
3. You can adjust the amount. A good heavy rain setting is **3**.
4. Now, change the color to a dark blue or grey. If left white, it can be mistaken for snow quite easily.
5. Finally, adjust the puddles amount to **5** and the fresnel to **4**; this will give a dramatic yet realistic looking wet effect.

This type of entity is best suited for large, out-of-door types of settings.

How it works...

The *Post Effect* entity renders on all objects and players in its radius. The underside of objects will remain dry, and objects can be adjusted in real time to show how the rain runs down slanted surfaces.

The advantage of this entity over the simple space loop particle effect or using the wet layer on objects is that all objects within the radius of the entity are affected and can be changed or dynamically modified in real time. Objects can even be half in and half outside of the light's radius. Crytek developed a genius solution in CryENGINE 3 where areas underneath objects that would be obstructed by "real" rain are indeed occluded of the wet effect. The full screen rain effect will automatically disable if objects are seen in screen space to be above the player that would occlude the rain.

There's more...

You may have noticed some other *Atmospheric* entities available to you. The following explains some of the ones that pertain directly to a stormy or rainy environment.

Lightning entity

Add the *Lightning* entity to your level from the *Entity/Environment* section. This will enhance the rain and storm effect by adding a procedural particle and light effect, which can be adjusted to suit most situations.

Apply the **sounds | environment:fantasy:distant_thunder_oneshot** to **hear a thunder sound** after every lighting effect. Then edit the delay to simulate the distance to the effect.

Wind

The global wind speed can be adjusted by entering the numbers (values in meter/second) into the **Environment** settings wind vector field (positive y values determine fast wind to the north, negative y to the south, positive x to the east, and negative x to the west).

This can be important as all plants and entities that have wind bending properties will react to this and can be used to simulate strong storms.

Fog volumes

The *Fog* entity is used to overwrite the fog effect in specific areas. In the indoor areas, a subtle fog effect can be added to simulate a dusty area. To use, create an *Area Box* and attach the *Area Box* to the *Fog* entity. When in game mode, you can walk into the area and see how the fog gets different inside. Outside the game mode, the fog is not visible.

5

Basic Artificial Intelligence

In this chapter, we will cover:

- ▶ Placing the enemy AI
- ▶ Generating the AI navigation
- ▶ Forbidden boundaries
- ▶ Forbidden areas
- ▶ Setting up the interior navigation
- ▶ Debugging the AI triangulation
- ▶ Narrowing the AI's FOV to allow attacks from behind
- ▶ Respawnning AI

Introduction

Realistically rendered and animated characters require state of the art AI systems to intelligently respond to the game environment and maintain the illusion of realism. CryENGINE 3 Sandbox™ lets designers control basic AI using the flow graph visual scripting system, placing most AI gameplay control in their hands.

After creating a small environment for your Player to run around in, it is now time to place down some AI for the Player to fight against. For this chapter, we will cover how you can utilize many of the basic AI systems that are already in place for the CryENGINE 3.

Placing the enemy AI

In this recipe, we will show how you may place down simple AI grunts from the Entities rollout for the player to fight against.

Getting ready

- ▶ Before we begin, you must have the Sandbox 3 open
- ▶ Then open `My_Level.cry`

How to do it...

1. In the **RollupBar**, click on the **Entity** button.
2. Under the **AI** folder, find **Grunt**.
3. With the **Follow Terrain** enabled, click-and-drag anywhere on your terrain.

How it works...

The *Grunt* is a simple AI entity that packages a whole lot of extra properties that we will touch on later in this chapter. The *Grunt* itself is a simplistic human AI that borrows a lot of its functionality from the basic actor defined in LUA, which means it shares a lot of the human characteristics which the player also utilizes, such as movement.

There's more...

The **AI can be placed as *Entity Archetypes***. **Although not essential, it is a very good practice** for the development team to create templates of enemies for the player to fight (a.k.a. *Archetypes*). These *Archetypes* will hold any of the properties such as **attackrange**, **FOV**, **Health**, and other **properties that designers want these AI to have across their entire game**. The *Archetype Library* can be found in the *Database view*.

See also

- ▶ The *Creating Prefabs to store in the external libraries* recipe

Generating the AI navigation

Even though you've placed an AI down on your level, the AI may not react to the player's presence when going into game mode. For that to happen, the AI needs some sort of Navigation for it to trigger its basic states. This recipe will show you how to generate the AI navigation.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`

How to do it...

1. On the top main toolbar, click **AI | Generate All Navigation**.

How it works...

As simple as this is, this step is essential for working with AI, due to the fact that this recipe needs to be repeated every time there is a change in the AI's navigation such as modifications to forbidden areas, forbidden boundaries, cover spots, smart objects, interior navigation, and so on. With that said, get used to regenerating the AI's navigation a lot.

There's more...

You can generate the AI triangulation by clicking on **AI | Generate Triangulation**.

Generating AI triangulation

This method of generating the AI navigation will only update the triangulation made on the terrain and interior navigation. If you are updating only the forbidden areas or boundaries, this generation may speed up the processing time of generating the AI navigation (especially if you might be working with 3D volume navigation alongside it).

See also

- ▶ The *Forbidden boundaries* recipe
- ▶ The *Forbidden areas* recipe
- ▶ The *Setting up the Interior Navigation* recipe
- ▶ The *Using the interior AI navigation points* recipe
- ▶ The *Debugging the AI triangulation* recipe

Forbidden boundaries

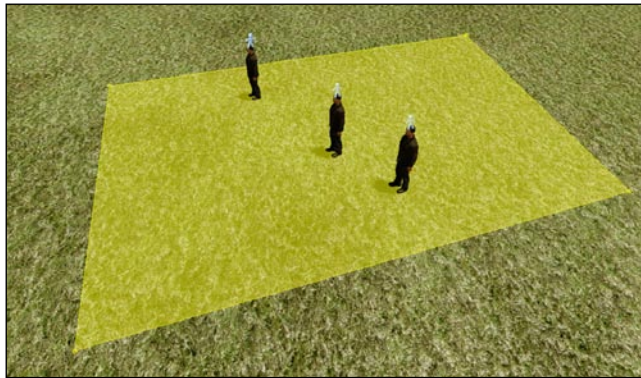
In this recipe, we will demonstrate how you may be able to prohibit an AI from crossing an invisible boundary by basic movement. Much like a child's play pen or a dog's yard, you will be building a fence in an area for your AI.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Place three AI Grunts roughly five meters apart in any fashion

How to do it...

1. In the **RollupBar**, click on the **AI** button.
2. From the **Object Type**, select **ForbiddenBoundary**.
3. With the **Follow Terrain** enabled, click four points roughly in a square around the placed Grunts.
4. Generate the AI navigation.



How it works...

The forbidden boundary is much like a fence for the cattle on a farm. The forbidden boundary deters the AI entities from crossing it with their basic movement behaviours, but they may still be able to cross it if they use attacks or other special animations that change their location at the end of the animation (for example, jumping a fence using *Smart Objects*). Once crossed, they will not attempt to try and return to the other side.

Remember to regenerate the AI after creating or changing a forbidden boundary.

See also

- ▶ The *Forbidden areas* recipe

Forbidden areas

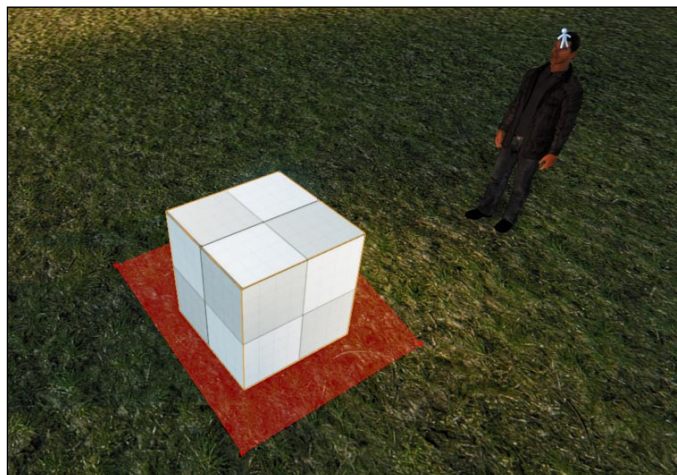
Where forbidden boundaries fail to keep the AI from going into certain locations, forbidden areas are used instead. This recipe will teach you how to utilize the forbidden areas to block off the areas that the AI entity should not go to.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Place down one Grunt AI
- ▶ Place `default\primitive_cube.cgf` as a brush down near the Grunt AI

How to do it...

1. In the **RollupBar**, click on the **AI** button.
2. From the **Object Type**, select **ForbiddenArea**.
3. With the **Follow Terrain** enabled, click four points roughly in a square around the **primitive_cube.cgf**.
4. On the **Forbidden Area Params**, set **DisplayerFilled** to true.
5. Generate the AI navigation.



How it works...

You may have noticed that without the forbidden area present around the box, the AI Grunt takes a significantly longer time to find a path to the player and may even get stuck running up against it if the player is behind the box. This is because the AI does not know of the box's existence and needs to be told to navigate around this area. Once the forbidden area is marked up around the box, the AI will have very little trouble navigating around the object to find the player.

Similar to forbidden boundaries, forbidden areas act as a stronger barrier against the AI from entering into them. If the AI manages to enter these forbidden areas, then the default AI behaviour, if active, will alert them to seek the nearest edge of the forbidden area to exit the area as quickly as they can.

See also

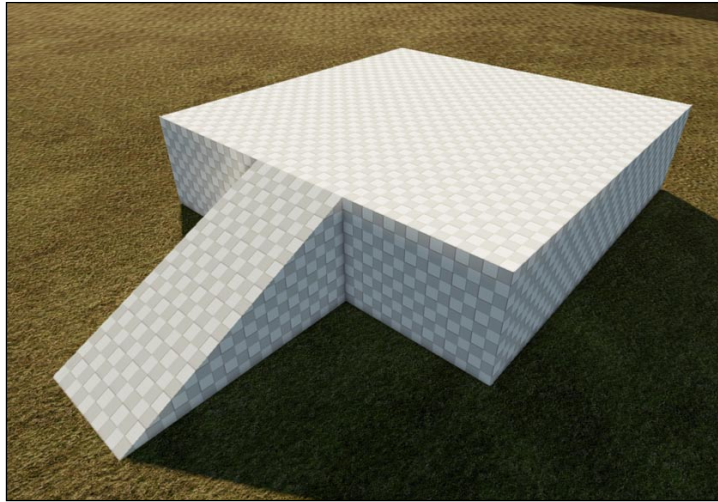
- ▶ The *Forbidden boundaries* recipe
- ▶ The *Debugging the AI triangulation* recipe

Setting up the interior navigation

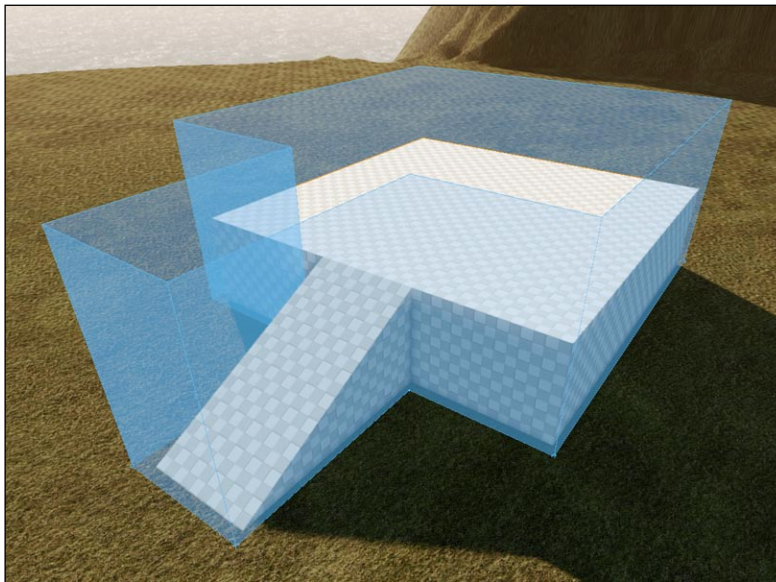
In this recipe, we will demonstrate how you may be able to get your AI to navigate on top of objects as well as interiors.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Build a platform from the *Solids* tool to the **dimensions of** `20 x 20 x 5`
- ▶ Again, using the *Solids* tool, build a ramp that leads up to the platform



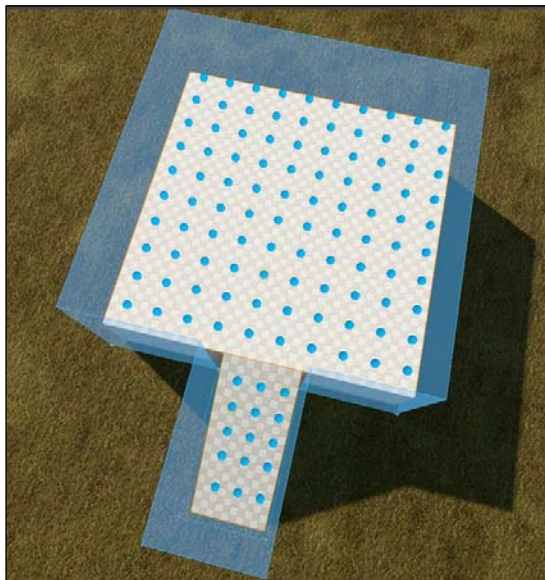
- Also, be sure to cover the ground section around the solid with a forbidden area (see the *Forbidden areas* recipe)



How to do it...

AI Navigation Modifier: Before we place down any of the nodes required for navigation, we must first outline the area that we want the AI to perceive as an interior navigation. We do this by creating an area around the object known as an *AI Navigation Modifier*.

1. In the **RollupBar**, click on the **AI** button.
2. From the **Object Type**, select **AINavigationModifier**.
3. With the **Follow Terrain** enabled and **Grid Snap** set to **1 meter**, click eight points that surround the platform and the ramp (remember to double-click on the eighth point to finalize your area).
4. In the **AINavigationModifier Params**, change the **Height** value to **10** (meters).

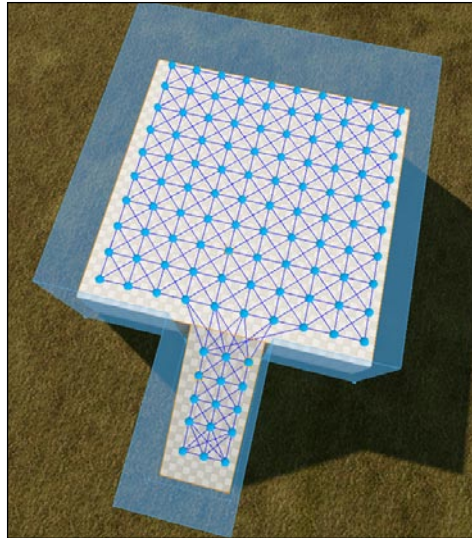


5. In the **AINavigationModifier Params**, change the **WaypointConnections** to **Auto-Dynamic**.

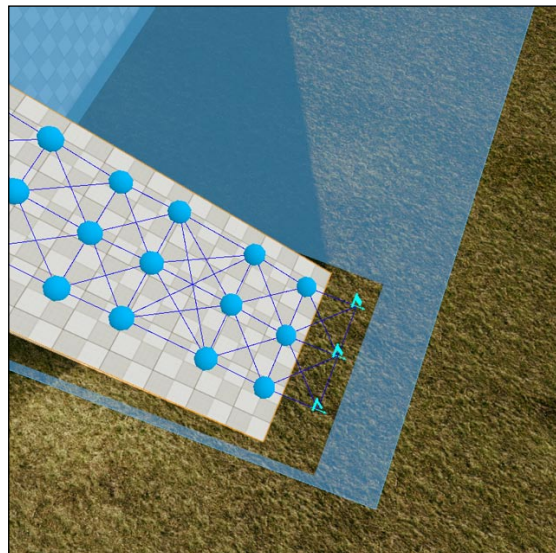
AI Points: Now that we've created an interior area for the AI to recognize, we must now create the navigation points within the area defined for the AI to use as the navigation triangulation. To do this, we must create the *AI points*.

1. In the **RollupBar**, click on the **AI** button.
2. From the **Object Type**, select **AIPoint**.
3. With **Follow Terrain** and **Snap to Objects** enabled, and the **Grid Snap** set to **1 meter**, click **1** AI Point into a corner at the top of the platform.

4. Copy and paste 99 additional points covering the whole top surface with a two meter gap between each point (10 points x 10 points).
5. Do the same with the ramp by covering the surface with 15 AI **points** (3 points x 5 points).

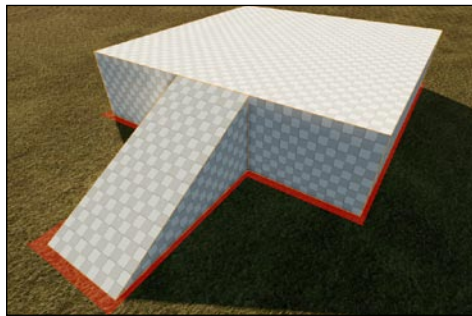


6. To connect all the points, you must now generate the AI navigation. See the *Generating the AI navigation* recipe. You should now see all of your interior navigation points connected.



Entry/Exit Points: Before we can get the AI to travel between the Interior Navigation and the Terrain Navigation, we must first create Entry/Exit Points from the AI points that are placed onto the Terrain, but also within the Navigation Modifier.

1. Copy three of the AI points on the lowest end of the ramp and place them onto the Terrain two meters away from the lowest AI points.
2. Selecting each point one by one, change their **Type** from **Waypoint** to **Entry/Exit**.
3. Select the **AINavigationModifier** and **Edit Shape** to move the two points at the end of the ramp to come out a bit farther to wrap around the three entry/exit points.
4. Generate the AI navigation.



How it works...

The *Interior Navigation* is a node-based AI triangulation tool. This tool allows for specifically placed nodes to act as vertex points for the break up of the triangulation. Using the interior navigation nodes also allows for 3D navigation to be possible, so that the designers are able to create land bridges to allow AI to travel across the top as well as underneath the bridge. This method is used in cases where the object that the AI needs to travel on is roughly two meters higher than the terrain triangulation.

There's more...

Here is some additional information about the interior navigation points that you should know.

Auto-Dynamic Points versus Designer Controlled Points

The main difference between the generation of the AI points for the interior navigation is that the *Designer Controlled Points* are AI points that can manually be linked by the designer. It usually takes a long time to link these points up, but if – for whatever reason – the designer needs to have total control over how the links operate, then this is possible.

However, *Auto-Dynamic* automatically links the nodes if they are within the range of another node (defined by *NodeAutoConnectDistance*) and is not obscured by the edge of the *AINavigationModifier*.

See also

- ▶ The *Forbidden areas* recipe
- ▶ The *Debugging the AI triangulation* recipe
- ▶ The *AI Anchors* recipe

Debugging the AI triangulation

In this recipe, we will be showing you how you can debug the AI's triangulation through certain cvars.

Getting ready

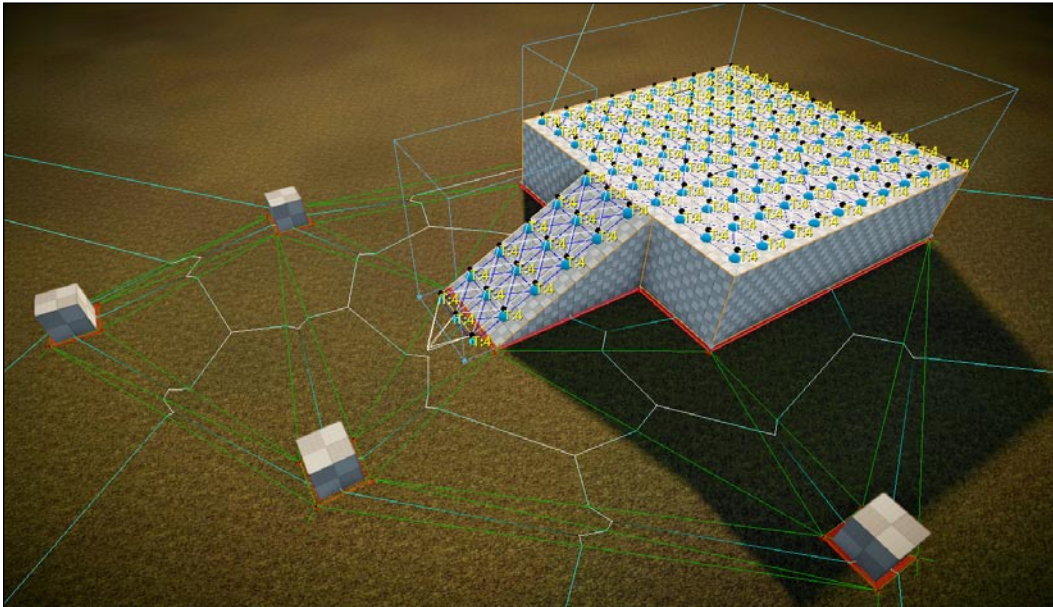
- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Complete the previous recipes with the basic AI

How to do it...

- ▶ `ai_debugdraw 1`: Shows the basic information for all AI within the level (*Behaviors*, *Actions*, *Target*, *Stance*, and so on).



- ▶ `ai_debugdraw 74`: Shows both the interior navigation nodes as well as the terrain triangulation with the current AI navigation information (generate the AI navigation to update this information).



How it works...

Utilizing both the debug methods will allow the designers to identify where the AI is going and how the AI is utilizing the triangulation that was generated within the level.

See also

- ▶ The *Placing the enemy AI* recipe
- ▶ The *Generating the AI navigation* recipe
- ▶ The *Forbidden boundaries* recipe
- ▶ The *Forbidden areas* recipe
- ▶ The *Setting up the interior navigation* recipe

Narrowing the AI's FOV to allow attacks from behind

In this recipe, we will be showing you how you may be able to manipulate the AI's primary and secondary FOVs to allow the player to have an easier time to sneak up on the AI from behind.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Complete the *Placing the Enemy AI* recipe

How to do it...

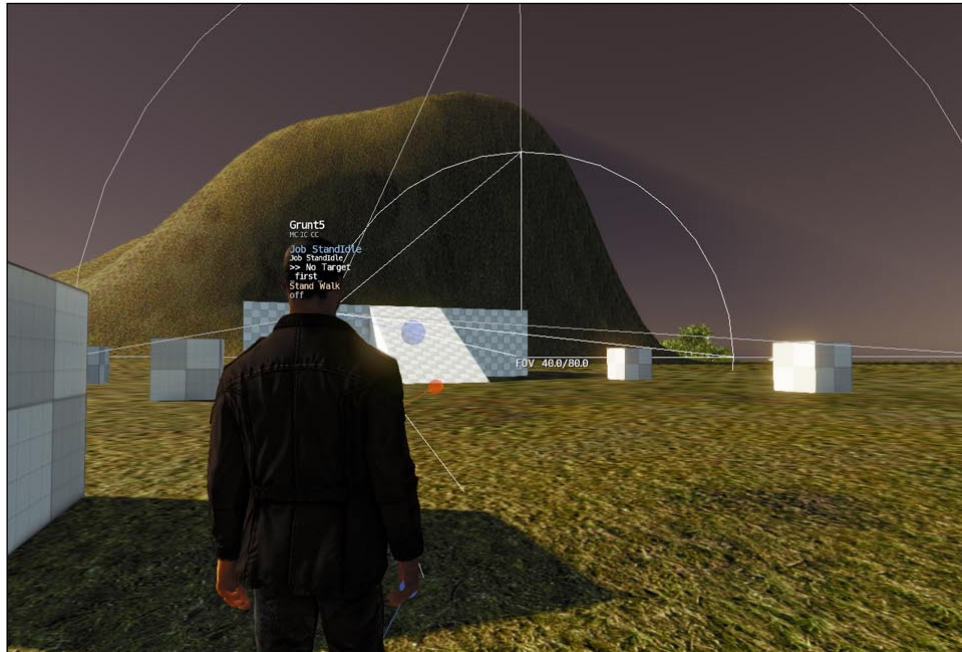
1. Change the following entity parameters on the AI Grunt that is placed:
 - ❑ **awarenessOfPlayer** = 0
2. Perception sub rollout
 - ❑ **FOVPrimary** = 40
 - ❑ **FOVSecondary** = 80

How it works...

- ▶ **awarenessOfPlayer** handles if the AI knows where the player is at all times. Making sure this value is false makes the AI do perception checks to see if it can find the player.
- ▶ **FOVPrimary** handles the main FOV of the AI and uses a quick reaction system if the player is caught within this FOV.
- ▶ **FOVSecondary** handles the peripheral vision, which uses a slower reaction system to investigate the location of where the AI spotted suspicious activity within that area.

There's more...

ai_DrawAgentFOV = 1 allows the designer to see the AI's FOV and sight range within the viewport (this must also have **ai_DebugDraw = 1** turned on as well).



See also

- ▶ The *Placing the enemy AI* recipe

Respawning AI

In this recipe, we will explore how you will be able to utilize the Territory and Wave systems along with a little bit of FlowGraph scripting to make your AI respawn after a couple of seconds of death.

Getting ready

- ▶ Before we begin, you must have Sandbox 3 open
- ▶ Then open `My_Level.cry`
- ▶ Place down one Grunt within the level

How to do it...

Making a territory:

1. From the **RollupBar**, open the **Objects** tab and click the **Entity** button.
2. Open the **AI** folder and select **AITerritory**.
3. Click down four points to surround the AI Grunt (double-click to finalize).

Placing the AI wave entity:

1. From the **RollupBar**, open the **Objects** tab and click the **Entity** button.
2. Open the **AI** folder and select **AIWave**.
3. Place the AI wave entity close to the **AITerritory**.

Linking the territory to the wave:

1. Select **AITerritory1** and click the **Pick** button.
2. Click **AIWave1** to pick that entity as the target.

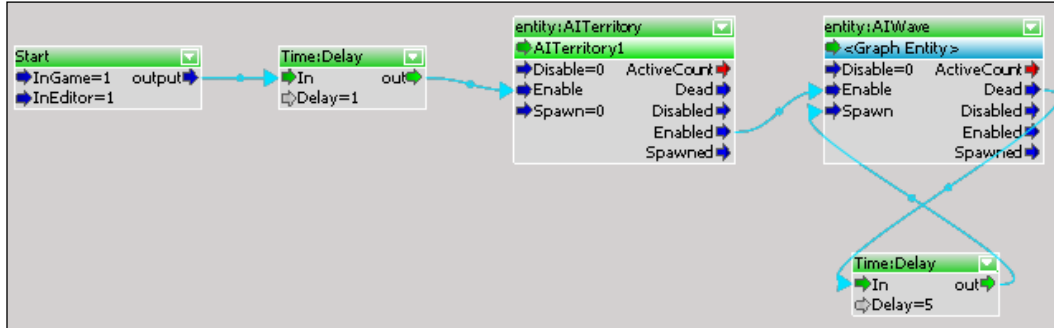
Giving the AI Grunt the territory and wave IDs:

1. Select the Grunt.
2. Under **Entity Properties2**, assign **AITerritory1** as the territory and **AIWave1** as the wave.

Creating a simple **FlowGraph (FG)** to respawn the Grunt:

1. Right-click **AIWave1** | **Create Flow Graph**.
2. Click **OK**.
3. Select **AIWave1** from **Viewport**.
4. In **FG**, right-click | **Add Selected Entity**.
5. Select **AITerritory1**.
6. In **FG**, right-click | **Add Selected Entity**.
7. In **FG**, right-click | **Add Node** | **Misc** | **Start**.
8. In **FG**, right-click | **Add Node** | **Time** | **Delay**.
9. Copy and paste the **Time:Delay** node and set the **Delay integer** on the second node to **5** (seconds).
10. Link (click-and-drag) **Start output** to **In** on **Time:Delay 1** second.
11. Link **out** from **Time:Delay 1** to **Enable** on **AITerritory1**.
12. Link **Enabled** from **AITerritory1** to **Enable** on **AIWave1**.
13. Link **Dead** from **AIWave1** to **In** on **Time:Delay 5**.

14. Link **out** from **Time:Delay 5** to **Spawn** on **AIWave1**.



How it works...

With the AI territories and wave system, we can link this into a flow graph to allow the AI within the territory and wave to respawn after five seconds of their death. Anything outside of the AI wave will not respawn.

There's more...

Here are some definitions for the AI territory and wave FlowGraph nodes that you should know about.

AI territory FlowGraph node properties

Input:

- ▶ **Disable:** This disables the territory
- ▶ **Enable:** This enables the territory
- ▶ **Kill:** This kills all active AIs of this territory
- ▶ **Spawn:** (Re-)Spawns AIs assigned to this territory (but not assigned to any wave)

Output:

- ▶ **ActiveCount:** Outputs the number of currently active AIs assigned to this territory
- ▶ **Dead:** Fires when all AIs assigned to this territory are dead (works also for respawned AIs)
- ▶ **Disabled:** Fires when the territory gets disabled (it doesn't fire on level start, although all territories are disabled from start automatically)
- ▶ **Enabled:** Fires when the territory gets enabled
- ▶ **Spawned:** Fires when AIs are spawned via the spawn input (if the territory was disabled before, it gets enabled and the enabled output fires as well)

AI wave FlowGraph node properties

Input:

- ▶ Disable: This disables the wave
- ▶ Enable: This enables the wave
- ▶ Kill: This kills all active AIs of this wave
- ▶ Spawn: This (re-)spawns AIs assigned to this wave

Output:

- ▶ ActiveCount: Outputs the number of currently active AIs assigned to this wave
- ▶ Dead: Fires when all AIs assigned to this wave are dead (works also for respawned AIs)
- ▶ Disabled: Fires when the wave gets disabled (it doesn't fire on level start, although, all waves are disabled from start automatically)
- ▶ Enabled: Fires when the wave gets enabled
- ▶ Spawned: Fires when AIs are spawned via the spawn input (if the wave was disabled before, it gets enabled and the enabled output fires as well)

6

Asset Creation

In this chapter, we will cover:

- ▶ Where to install the CryENGINE 3 plugin for 3ds
- ▶ Creating textures using CryTIF
- ▶ Setting up units to match CryENGINE in 3ds
- ▶ Basic material setup in 3ds
- ▶ Creating and exporting static objects
- ▶ Creating and exporting destroyable objects
- ▶ Using advanced material editor parameters to create animation
- ▶ Creating new material effects
- ▶ Creating image-based lighting

Introduction

In this chapter, we will explore the fundamentals, as well as some advanced techniques, to creating real-time game assets for the CryENGINE 3.

Throughout this chapter, we will use 3dsMAX2010 and all examples will assume a basic understanding of that package.



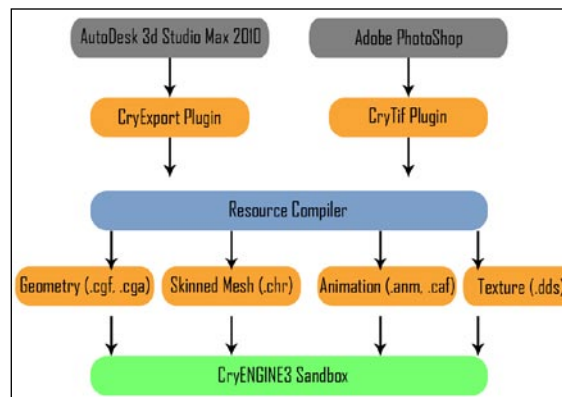
CryENGINE 3 also supports Maya; see mycryengine.com for more details.

The CryENGINE 3 using a system that considers two types of assets:

- ▶ Source assets, which include .TIF format textures and any other uncompressed information
- ▶ Target assets, which include .DDS and other compressed data and may be specific to the target platform (Xbox, PS3, and others)

The handling of these different file types is by the resource compiler, which is explained in *Chapter 1, CryENGINE 3: Getting Started*.

We will follow the overall explanation of the asset pipeline for CryENGINE by exploring each step from start to finish.



Installing the CryENGINE 3 plugin for 3D Studio Max

As we will be using 3D Studio Max 2010 in the asset creation examples of this book, it is important that we explore the installation of the plug-in for 3ds max to allow for the exporting of meshes, animation, and other data to CryENGINE.

This first step in creating assets for the CryENGINE 3 is an important one. As most assets used within the CryENGINE need to be in an optimized format, we must export all of the source assets (asset we will work on in external packages like 3d studio max and Photoshop) through an optimization and conversion process. For this process to be successful it is imperative that the **RESOURCE COMPILER** be setup properly.



The SDK installer should automatically install the plugins for the installed versions of 3ds Max during the CryENGINE 3 tools installation. In this case, the installer automatically sets up the system path, which allows the exporter to communicate with the Resource compiler.

Getting ready

To prepare yourself for asset creation, you must have the CryENGINE 3 SDK installed.

Before copying the plugin to the required 3ds Max directory ensure 3ds max is shutdown.

Locate the following .dlu file that is matched to the 3d studio max version you will be using. These files are found in the /tools directory of your build.

- ▶ 3ds Max 9 32 bit use CryExport9.dlu
- ▶ 3ds Max 9 64 bit use CryExport9_64.dlu
- ▶ 3ds Max 2008 32 Bit use CryExport10.dlu
- ▶ 3ds Max 2008 64 Bit use CryExport10_64.dlu
- ▶ 3ds Max 2009 32 Bit use CryExport11.dlu
- ▶ 3ds Max 2009 64 Bit use CryExport11_64.dlu
- ▶ 3ds Max 2010 32 Bit use CryExport12.dlu
- ▶ 3ds Max 2010 64 Bit use CryExport12_64.dlu

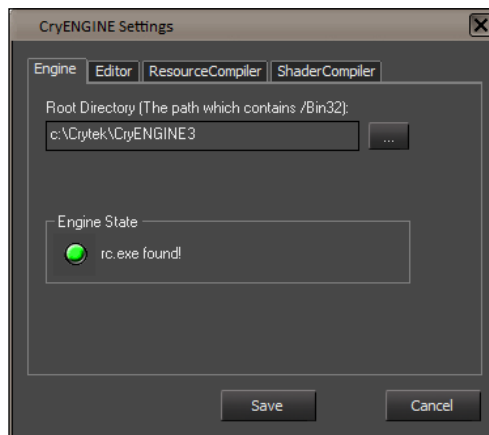
How to do it...

1. Once you have located the .dlu file that is associated to your installation of 3ds Max, copy this file to the plugins directory of your 3ds Max directory.

C:\Program Files\Autodesk\3ds Max 2010\plugins\CryExport12_64.dlu is the installation path and we will use 3ds Max2010 x64 bit for the examples in this book.

2. Open 3ds Max.

When the plugin initializes, you will be presented with a setup screen to direct the exporter from 3d studio max to the resource compiler in your installation.



3. In our example, we have our installation located in:
`C:\Crytek\CryENGINE3`
4. On clicking **Ok** you will now be able to access the exporter through:

Utilities | More | CryENGINE Exporter

How it works...

Now that we have installed the CryENGINE 3 exporter for 3ds and directed it to our resource compiler, we can export a variety of assets and objects to a CryENGINE format ready for use in Sandbox.

There's more...

There are a host of tools available to 3ds Max users that are shipped within the CryENGINE SDK. Find out more later on how to install and activate these extended tools, and what they are used for.

3ds Max CryTools Maxscripts

During installation, the CryTools Maxscripts may have been installed to your installation of 3ds Max depending on the settings used during installation. These CryTools scripts are a collection of Maxscript tools that were developed by various artists throughout the development of Crysis and over various iterations of the engine. These are all coded in Maxscript format and thus are only compatible with 3ds Max.

The tools are separated into logical areas of:

- ▶ Animation
- ▶ Artist
- ▶ Morph
- ▶ Rigging

Installing the 3ds Max CryTools Maxscripts

If the SDK installation does not install the CryTools Maxscripts, they can be manually installed. To install them simply copy the `LoadCryTools.ms` located under the `/tools/CryMaxTools` to the `Scripts/Startup` of the 3ds Max installation directory.

Uninstalling the 3ds Max CryTools Maxscripts

To uninstall the tools, delete the `LoadCryTools.ms` file located in the `Scripts/Startup` folder of the 3ds Max directory.

See also

- ▶ Having installed the required tools, you may want to go straight to the *Creating and exporting static objects* recipe later in this chapter
- ▶ You may also want to re-run the installation of the CryENGINE SDK, which is covered thoroughly in the *Installing the CryENGINE SDK* recipe in *Chapter 1, CryENGINE 3: Getting Started*

Creating textures using CryTIF

The textures we will create throughout the course of this book will be created in Photoshop and will be stored in the TIF image format saved using the CryTIF plugin.

CryTIF is a Photoshop plugin developed by Crytek that can load and save merged Photoshop images as .TIF files. When saving the .TIF file, the plugin calls upon the Resource Compiler to display a dialog to the user where the compression settings can be selected. The settings that get chosen in the dialog are stored as metadata on the TIF file.

It's important to realize though that the .TIF format images are not used directly in the engine but are rather converted to a more optimized format, in this case, from a .tif to a .dds, by the Resource Compiler.

Getting ready

Let's go over what you need to do to be able to create textures for CryENGINE.

Photoshop must be closed for the installation of the CryTIF plugin to work.

You should have completed tutorial one or have run the `SettingsMgr.exe` to ensure that the build path is set up correctly to your installation of the CryENGINE 3. You can find the `SettingsMgr.exe` in the root directory under `/Tools`.



The SDK installer should automatically install the plugins for the installed version of Photoshop during the CryENGINE 3 tools installation.

In case a manual installation is required, copy the following files to the root Photoshop directory:

- ▶ `Bin32\zlib1.dll`
- ▶ `Bin32\jpeg62.dll`
- ▶ `Bin32\libtiff3.dll`

Copy the file that enables support for the CryTif format

Tools\CryTIFPlugin.8bi to the root Photoshop \Plugins folder.

How to do it...

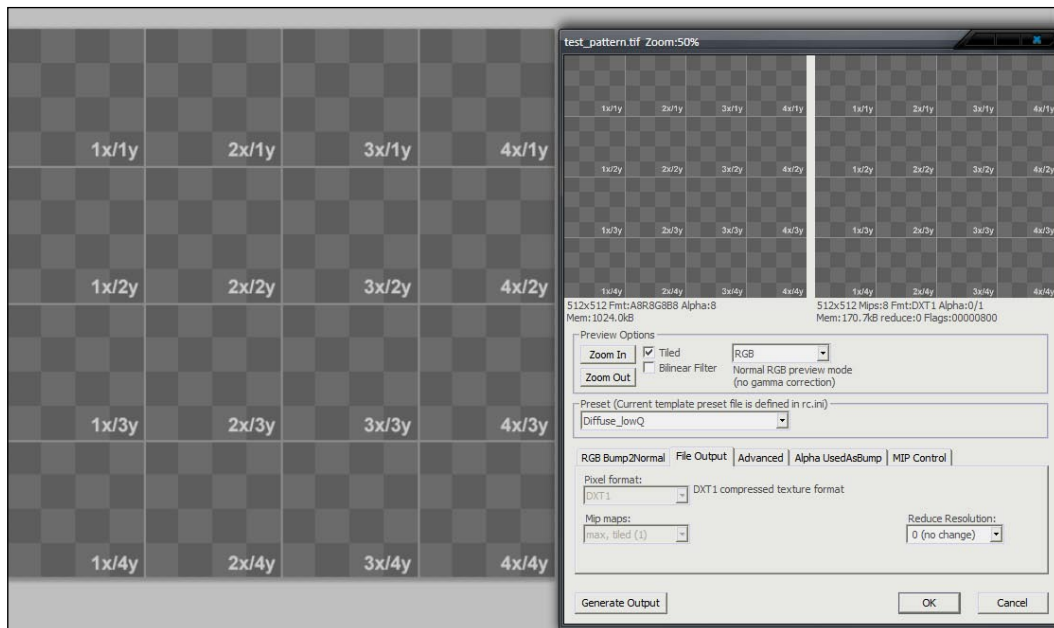
Let's go ahead and create our first texture:

1. Start Photoshop.
As we will simply be creating our first .TIF file, a complex texture is not required.
2. Create a new image with dimensions 512 x 512.
3. Create a simple pattern or import your own texture.



If your texture has an alpha channel on it, the CryTIF plugin will detect this and change its conversion process automatically.

4. Next, select **File | Save As** in Photoshop.
5. Save this file as a CryTIF (.TIF) file type. This format should now be available as a file format in the Photoshop file dialog.
6. Save this texture to your root directory under `textures/test_pattern.tif`.



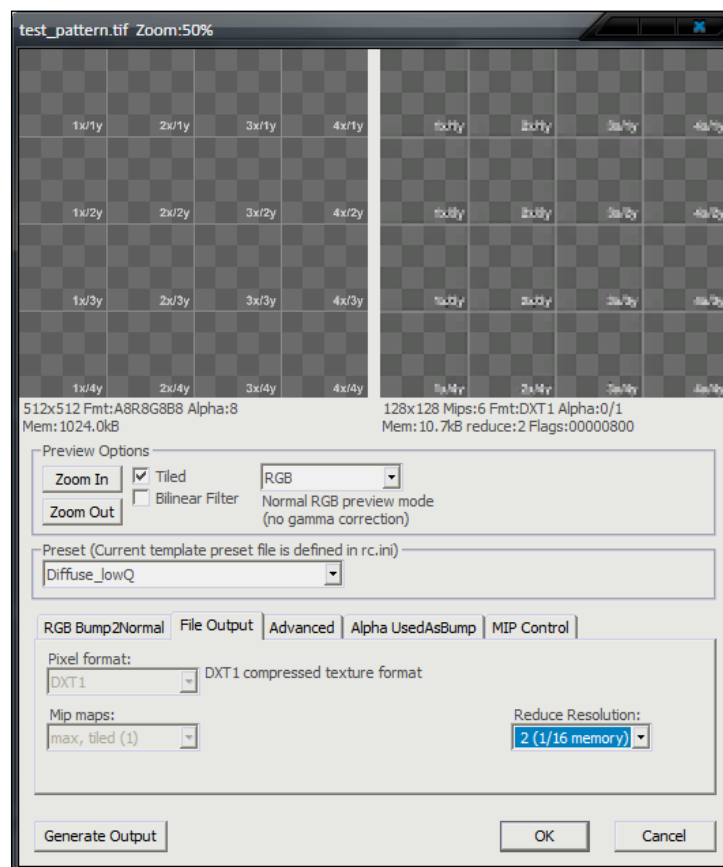
Once you select **Save** you will be presented with the CryTIF texture dialog. The primary function of this dialog is to add metadata to the texture, which can later be used by the Resource Compiler to compress the texture appropriately for whatever platform it is being created for.

In the upper area, you can see two images: the original image on the left, and a preview of the processed image using the preset on the right.

Directly below the two images are the different image properties such as resolution, number of mip maps, texture format, and memory consumption.

7. In this example, take the preset `diffuse_lowQ`.
8. Click one of the image previews and use your mouse wheel to preview the tiling of the texture.
9. Finally, notice the effects of reducing the resolution in this image by setting the reduce resolution to 2 (1/16th memory).

The preview image updates to show the compression that will be applied to the texture when conversion is performed:





Reducing resolution can drastically improve performance but can degrade the visual appearance.

10. Set the reduce resolution back to 0.
11. Click on **Generate Output** to manually perform the .TIF to .DDS conversion.
Pressing the **Generate Output** button forces an immediate conversion to the target output format without closing the CryTIF dialog box.
12. Click **OK** to write the meta data to the .TIF file and to save the .TIF, which is now ready for use in CryENGINE!

How it works...

When applying a texture to a material in CryENGINE using the .TIF extension, it will automatically load the .DDS version. For example, a diffuse texture path of `textures/test_pattern.tif` will load the file `textures/test_pattern.dds` at runtime.

The CryTIF (.TIF) format is used in the overall asset creation pipeline as an intermediate file format. The .TIF format is used because it contains the uncompressed texture information as well as having the ability to store different settings for the Resource Compiler as meta information.



Creating .dds files manually or without the Resource Compiler is not recommended.

There's more...

The CryTIF plugin is a very artist-friendly tool, but it can be even more powerful when the presets and the .dds file generation are completely understood. These are discussed further as follows.

Editing the CryTif plugin root path

Should an error occur and the system path to the root directory is lost by the plugin, you can restore the path Photoshop under `Help/About Plug-In/CryTIFPlugin`.

If the RC executable cannot be found, the configuration dialog will be opened automatically and a path to your root directory can be entered.

Adjusting the default presets in the CryTIF dialog

This drop-down list holds the presets that the texture can have saved onto it for use in conversion. It is important to select an appropriate preset for the type of texture you are creating, as not all textures can use the same settings; normal maps, for example, require a different compression than diffuse maps. Some common presets used are Diffuse_highQ for a diffuse texture, Specular_HighQ for specular, and Normalmap_highQ for Normalmaps. The preset with _lowQ produces slightly lower quality results at the benefit of less memory consumption in some cases.

You can add new presets to the `.ini` file by locating the file in the `bin32/rc/`. By default, the resource compiler uses the `rc_presets_pc.ini` file.

Manually generating the .DDS output

As the .DDS file is used as the final output, it is possible to edit the source .TIF file and then switch to CryENGINE and look at the results of the settings that you have changed instantly. The resource compiler automatically reloads/updates changed textures when Sandbox is running. This is commonly used for quickly testing compression settings without reopening the texture dialog box with every new conversion setting.

See also

- ▶ Now, having created a texture for use in CryENGINE, you can preview it quickly by jumping to the *Making basic shapes with solid tool* recipe in *Chapter 3, Basic Level Layout*. You can apply this texture to whatever solid you create
- ▶ You may also want to continue to the next recipe to set up your units in 3ds to be able to model your own objects to apply your texture to

Setting up units to match CryENGINE in 3ds

The CryENGINE 3 uses a metric scale to represent objects, distance, and size. It is important to keep this in mind when making photo-realistic assets. To make objects accurate to real-world measurements the unit setup in your DCC package must be set up to match the scale in the engine.

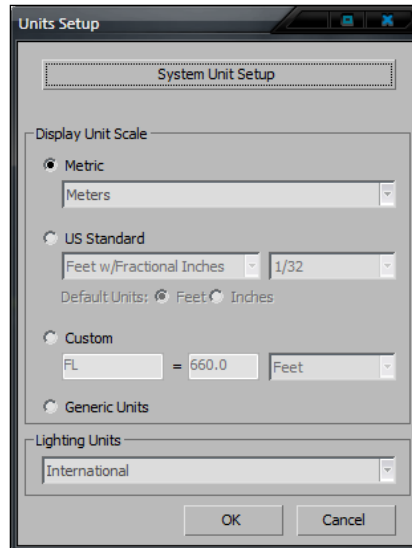
Getting ready

To prepare yourself for modeling objects for use in CryENGINE, 3ds Studio Max must be open.

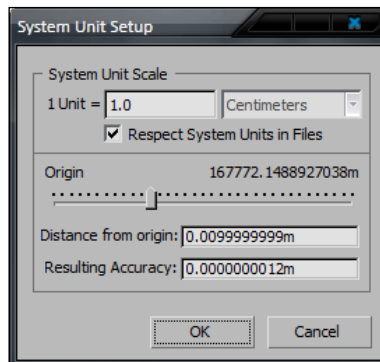
How to do it...

Different game engines use a different scale to represent objects; in our case let's set up our 3ds Max application to work with CryENGINE 3:

1. Along the top menu bar of 3ds locate **Customize**. Click it and select **Units Setup** from the list.



2. In **Unit Setup**, change the **Display Unit Scale** to **Metric Meters**.
3. Next, click **System Unit Setup**.



4. From here change your **System Unit Scale** to **1 Unit = 1.0 Centimeters**.
5. Click **OK** to go back into your 3DS Scene.



Depending on the size of your asset, you may require different grid settings for accurate representation of scale.

How it works...

Now that the scale and unit size have properly been set up, we can now accurately represent our model in 3ds Max the way it is meant to be rendered in CryENGINE.

It is common for some artist to load a scale human model when making some assets to maintain believability.

One of the major advantages of using metric with 3ds is that it makes all unit inputs and calculations, for example, when creating primitives, match the metric values.

There's more...

Unit setup can be a bit broader than just setting metric units. To learn about techniques used when working in metric scale within 3ds as well as some measurement references, continue to the following section.

Grid and Snap settings

Depending on the type of asset you may be creating, it is recommended to adjust the view port grid to a suitable setting.

Click the **Tools** menus at the top of 3ds and then go to **Grids and Snaps** and finally select **Grids and Snaps Settings**.

In these settings, you can change the Grid Spacing to other values. For large objects using a grid spacing of 1 meter is recommended. For smaller objects 1 cm is usually sufficient.

Measurement reference

The following reference measurements were originally created for the game Crysis, but are useful as a guide for creating new assets:

- ▶ Table Height (top of table) = 75 cm
- ▶ Seat height (top of chair, seat) = 46 cm
- ▶ Stairs height = power of 75 cm (150 cm, 225 cm, 300 cm, 375 cm, 450 cm)
- ▶ Stair steps = 18.75 cm (4 steps will sum up to 75 cm)
- ▶ Boxes = 60cm x 40cm x 40cm
- ▶ Hide Objects (which AI are prone to hide behind) = objects that are smaller than 120 cm

- ▶ Hide Objects (which AI crouch to hide behind) = objects that are bigger than 120 cm
- ▶ Hide Objects (which AI stand to hide behind and strafe left/right to leave cover) = objects that are bigger than 180 cm
- ▶ Jump over Objects (standing, without hand) = 30 - 50 cm high
- ▶ Jump over Objects (running jumps) = 50 - 100 cm high
- ▶ Jump over Objects (sideways, using hand) = 120 - 150 cm high

See also

- ▶ Having set up your units in 3ds max you should now continue to the basic material setup in 3ds recipe in this chapter, or skip to creating and exporting static objects for 3ds to immediately get your geometry into the Sandbox

Basic material setup in 3ds

Before using textures and the advanced shading on objects, the CryENGINE needs the object to have a material ready to be read by the engine. These materials are stored as .MTL files.

In this recipe, you will learn how to create a material using 3ds Max and translate that material's information into an MTL file.

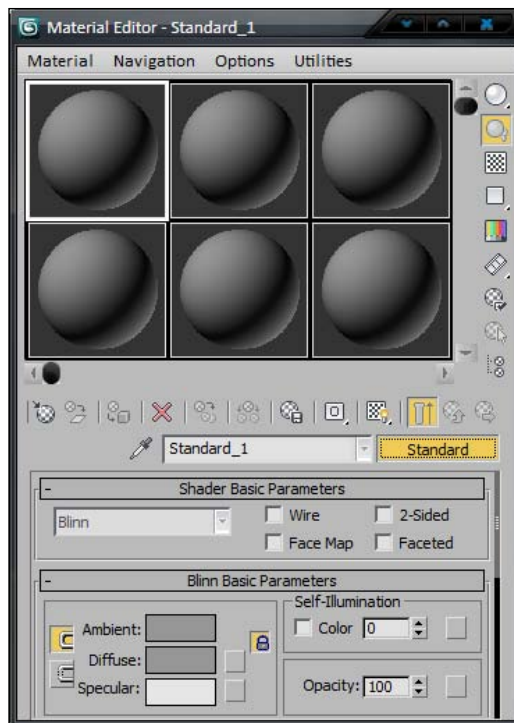
Getting ready

In this example, we will create a material for a basic prop with a texture and collision. To begin, open 3ds.

How to do it...

Let's set up our material for export to the engine:

1. Open the **Material Editor** in 3ds Max by pressing the keyboard shortcut *M*.
2. Create a new Multi/Sub-Object material in the first material slot by clicking the **Standard** button.



3. Select **Multi/Sub-Object** from the dialog that opens.
4. Next, set the number of sub materials to two by clicking the **Set Number** button in the newly created material.



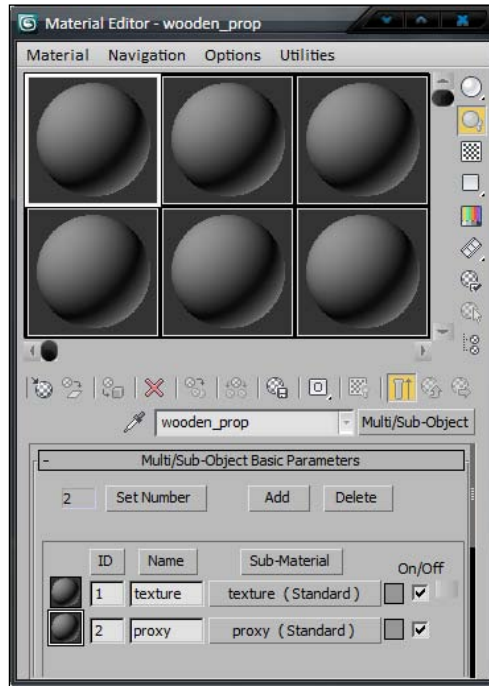
CryENGINE 3 supports up to 32 sub materials; however, every sub-material that is rendered will add to the overall draw call count of the asset. For the best performance keep the number of sub-materials low.

We should give the material and all sub-materials appropriate names. As and when we create the .MTL file, these names will be transferred and used by the engine.

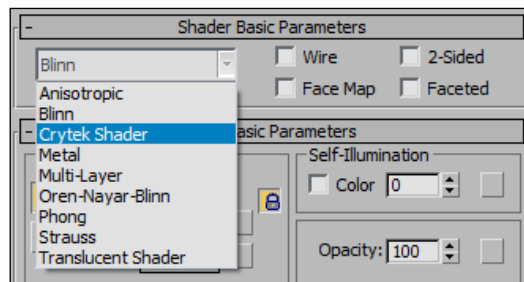


The name of the material that you assign to an object in 3ds Max must be the same as the name of the .MTL file created.

5. Name the parent material to **wooden_prop** and the subsequent sub-material to **texture** and **proxy**.



6. This next step is very important as each sub-material must be opened and the **Shader Basic Parameters** section changed to the **Crytek Shader** type.

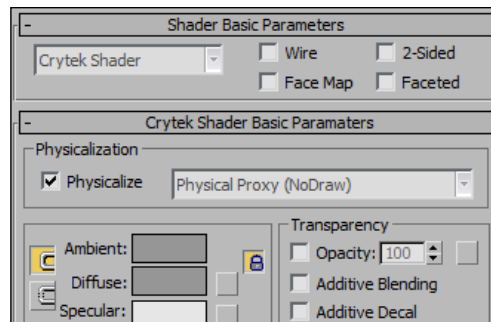



 Only use the **Crytek Shader** for the materials on objects.

Having selected the **Crytek Shader** for both submaterials, you would have seen a new parameter for these materials under the heading of Physicalization.

This setting controls the way the material acts in the engine, particularly with physics. The default setting can be used for most materials as it has no additional physics capabilities.

7. Select the **Proxy** sub-material and change its **Physicalization** type to **Physical Proxy (NoDraw)**. This will cause any geometry assigned this sub-material to be ignored by the renderer making it invisible to the user.
8. Next, to actually physicalize the material in the engine, select the **Physicalize** checkbox.

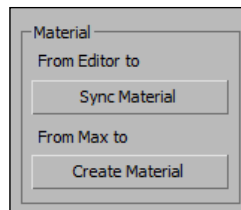


If **Physicalize** is disabled, the object will not physically interact with anything in the game world.

Now that our material is set up, we can create the .MTL file.

The .MTL file must be created in the directory you plan to save your assets in.

9. Open the CryENGINE export and locate the Material Rollout.



Ensure that the top-most material is active; you should see all the sub-material in their list. If a sub-material is active while creating the .MTL file, the file will contain only that sub-material.

10. Create the .MTL file by clicking on **Create Material**.

The **Sandbox Material Editor** window will open.

11. Click **Create Material** in 3ds Max a second time.
You will then, finally, be asked to enter a filename for the new MTL file.
Again ensure that this name is the same as the name of the material in 3ds Max.
12. For this recipe, save this material as `wooden_prop.mtl` to a folder named `objects/wooden_props/`
13. Click **Save** when completed.

How it works...

Multi/Sub-Object materials allow a mesh to have multiple material IDs on a per poly level. This means that two welded triangles can have different material IDs. This is useful many times when making complex objects.

The material has now been created and is readable by the engine. The `.MTL` files are `.XML` files that can be manually edited, if so desired.

Now that we have created this `.MTL` file, we can assign textures to it and use it for new or existing assets in the engine.

There's more...

You may want to know how to add textures to the 3ds max material now that you have created or got more information on the `Physicalize` parameter.

Assigning textures in 3ds Max to materials

You can assign textures to each submaterial, except for the Physical Proxy (NoDraw) submaterial which, as discussed, will not be rendered. The main three slots that are used for textures before generating or exporting an `.MTL` are the diffuse, specular, and bump slots.

Physicalize

When the *Physicalize* checkbox is checked, it actually stores information on the mesh for it to be physical. Simply setting a material to physicalize in 3ds and exporting it to engine will not cause it to be physical when applied to other objects. It is thus important to realize that adjusting the physical properties of collision will likely require a re-export of your mesh.

See also

- ▶ Having now created a material, you are ready to move onto the next recipe in this chapter, which teaches you how to create and export static objects
- ▶ Go back to the *Creating textures using CryTIF* recipe earlier in this chapter to create some textures to assign to your material

Creating and exporting static objects

Static geometry is used for various objects in the environment that do not have any special properties assigned to them. Some obvious examples could be walls, some structural components of levels, and other natural objects like large rocks.

Most of these would be considered static geometry because, typically, these objects would not need to have any advanced object setup to work.

In this recipe, we will make a typical wooden prop. As this tutorial is meant to show the process of exporting a basic mesh, it is not essential that you create an identical asset to the one created in this recipe.

Getting ready

This tutorial assumes that users understand the basics of 3ds Max, such as the user interface and the creation of simple geometry. It also assumes that the user can place objects into a level using the Sandbox Editor.



All directories containing objects must be placed under the root game folder in the Game\Objects folder. Objects placed outside of the Game folder won't work in the engine.

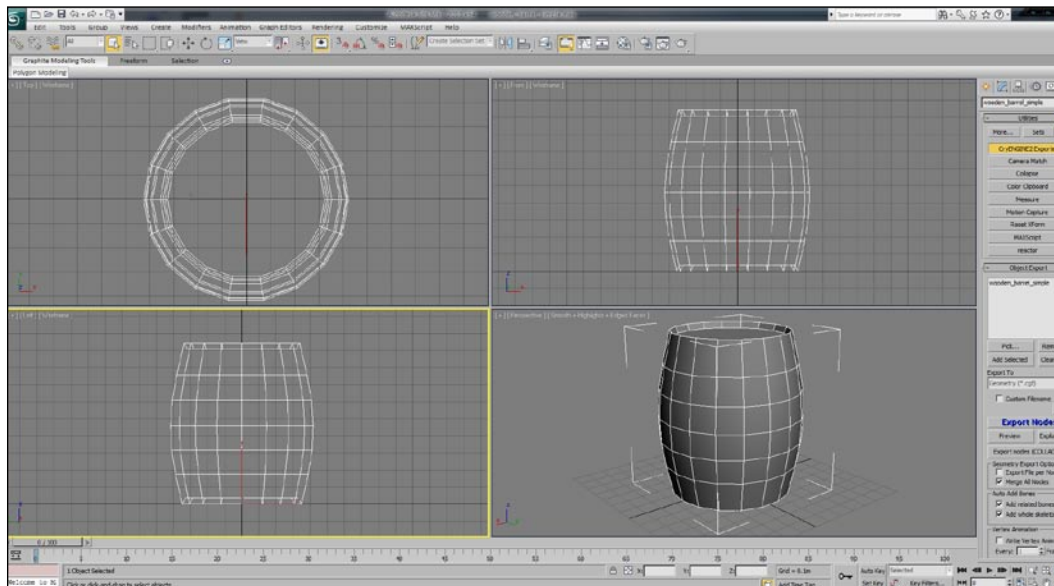
You should have just completed the *Basic material setup in 3ds* recipe previously and thus should have 3ds open.

How to do it...

Let's go ahead and get our hands dirty by making some models:

1. First let's create some geometry for your first object.

In this example, I have created a simple cylinder-type mesh that will become a wooden barrel. I have thus named the object or node in 3ds as `wooden_barrel_simple`:

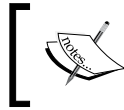


2. As the pivot of the object in 3ds Max will be the pivot of the object in the engine, align the pivot to the origin. This will make placing the object easier once it's in Sandbox.
3. Next, let's assign a material to our object. You can use the material we created in the previous recipe or create a new multi/sub-object material with two submaterials and name it `wooden_prop`.
4. To assign the `wooden_prop` material to the `wooden_barrel_simple` object, select the object and open the material editor.
5. Next, click on the **Assign Material to Selection** icon.



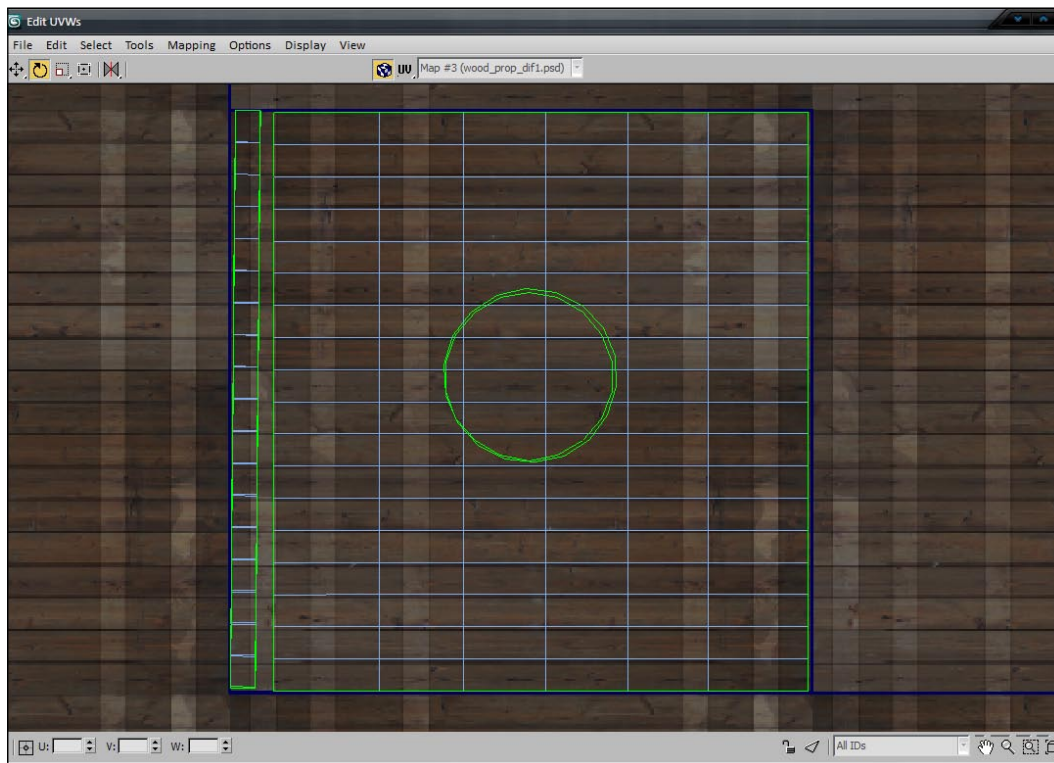
Now that we have our material applied, we can assign a texture.

6. Before assigning textures to our mesh, we must first make sure we have UV-mapped our object and that it has smoothing groups assigned. It is typically easier to preview smoothing groups with no texture being displayed.



During export, if smoothing groups are not found on the mesh, the exporter will generate them automatically, which can cause errors to appear on the mesh.

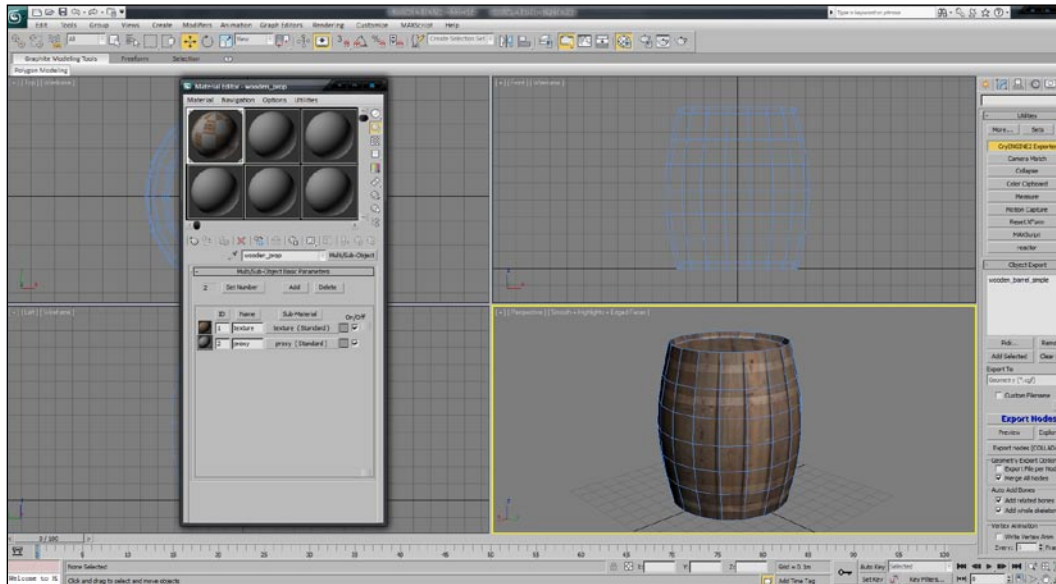
In this example, I created a simple cylindrical unwrap:



We can now assign a texture to the first submaterial. Ensure all the polygons in your object are set to material ID 1.

7. Assign an appropriate wooden-type texture to the material.

8. Finally, before exporting, it is important that your mesh be an editable poly or editable mesh and the modifier stack should be collapsed to avoid any unsupported modifiers on export.



9. Save your .MAX scene at the same location as you want to store the asset. In this case, save the .MAX scene to Game\objects\wooden_props\wooden_barrel_simple.max.
10. To export the object, open the 3ds exporter and locate the Object Export section at the top of the interface. Select your prop and click on the **Add Selected** button. You will notice its name is added to the export list.
11. Next, select the type of file you'd like to export. In this case, it is a geometry or .CGF file.
12. Next, click the **Export Nodes** button. This will export the object to the same folder as the 3ds Max file. By default, the exported object will be given the same name as the 3ds Max filename, in our case wooden_barrel_simple.cgf.

The final step of this process is to preview the asset in engine!

13. Open up the CryENGINE Sandbox and load a level.
14. On the rollup bar, select brush and browse to wooden_props/wooden_barrel_simple.cgf and drag it into the level.

You will see that the geometry is there but the texture is red and reads replace me.

This is simply telling us that our .MTL file does not have a valid texture in the diffuse slot.

15. To rectify this, select the prop and open the **Sandbox Material Editor**.
16. Click the *get material* from selected tool.
17. Then, browse to the diffuse slot of your object and using the browse button locate the texture you want to use for the object.



How it works...

To visualize objects in a world, CryENGINE uses the concept of render nodes and render elements. Render nodes represent general objects in the 3D engine. Among other things, they are used to build a hierarchy for visibility, allowing physics interactions, and finally for rendering. For actual rendering, these nodes add themselves to the renderer passing an appropriate render element, which implements the actual drawing of the object in engine. These objects are all seen as Y+ facing forward in the CryENGINE.

There's more...

You will definitely want to know more about optimizing the physics and collision of objects using a collision or proxy mesh. You can also learn about occlusion geometry and user-defined physics properties next.

Physics proxy

The physics proxy is the geometry that is used for collision detection. It can be part of the visible geometry or linked to it as a separate node. Usually the physics proxy geometry is a simplified version of the render geometry but it is also possible to use the render geometry directly for physics. However, for performance reasons, the collision geometry should be kept as simple as possible since checking for intersections on complex geometry is very expensive, especially if it happens often. A physics proxy is set up in the DCC tool exclusively. The only setup needed in Sandbox is assigning the surface type.

Occlusion geometry

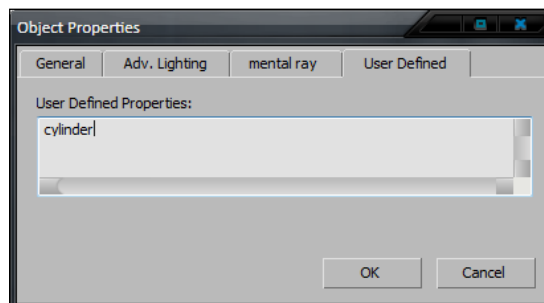
A simple geometry object that represents the shape of the render geometry is created in the DCC tool. It must be named "\$occlusion" and linked to the object to which it belongs. In our example, this is the wall piece. No special material setup is needed.

Important notes:

- ▶ Occlusion polygons will always be treated as double-sided in the engine. A simple single-sided plane in the DCC application might be a sufficient occluder for a solid wall.
- ▶ Occlusion geometry usually has to be rasterized on the CPU, which is a lot slower than drawing it on the GPU. So, keep the number of polygons of the occluder geometry as low as possible; for example, just use a single-sided plane for a complex wall.

User defined properties

It may be required that some objects have specific information or properties applied. This can be accomplished through using the **User Defined Properties** in 3ds Max. Select an object and on the **Edit** menu, click **Object Properties** to access the **User Defined Properties** of that object.



Some common parameters used are:

- ▶ `box` – Force this proxy to be a Box primitive in the engine
- ▶ `cylinder` – Force this proxy to be a Cylinder primitive in the engine

- ▶ `capsule` – Force this proxy to be a Capsule primitive in the engine
- ▶ `sphere` – Force this proxy to be a Sphere primitive in the engine
- ▶ `Mass = Value` defines the weight of an object `mass = 0` sets the object to unmovable

See also

- ▶ Having created and exported an object you can go to the *Utilizing Geom entities instead of Brushes* recipe in *Chapter 3, Basic Level Layout*
- ▶ Go to the next recipe in this chapter to learn how to create and export destroyable assets

Creating and exporting destroyable objects

There are several methods of setting up breakable assets that the player can destroy in CryENGINE. In this recipe, we will be going through the creation of a destroyable object.

Destroyable objects are assets that contain the original object and pre-created pieces that spawn when this original object is destroyed. It destroys into the pre-created pieces when taking more damage than the specified "health" property of the entity placed in sandbox as a destructible object.

Good candidates for destroyable objects are glass bottles, explosive barrels, computer monitors, and wooden barrels. Larger objects can also be destroyable, but it will depend mostly on what the game play calls for, such as a destroyable vehicle or wall.

All of these assets normally consist of pre-created pieces that emit when the object is destroyed.

Getting ready

You should have already created a basic object that you will in turn make destructible.

In this recipe, we will use the `wooden_barrel_simple` asset created in the last recipe.


How to do it...

To create a .CGF containing the object's broken pieces, we must create them as submodels for our main mesh.

Each submodel also must have physics proxy geometry. The submodel's name and its User Defined Properties determine its behavior.


The object must consist of the main original model.

This node or object MUST be named Main that will act as the alive geometry.

[ Main is the (only) pre-destruction submodel.]

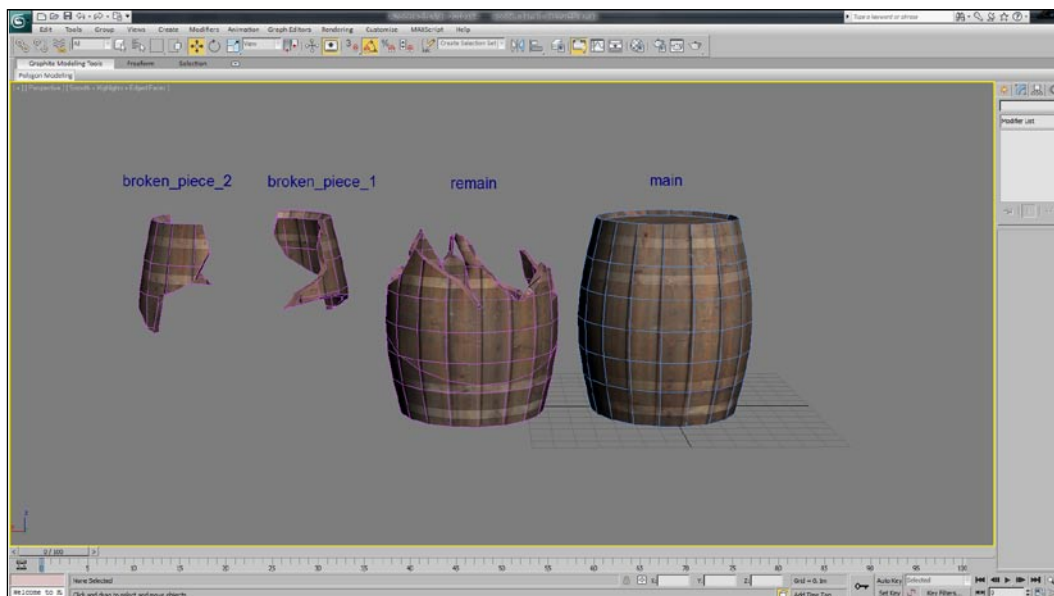
We must now create a new mesh that will be the destroyed geometry that will replace the original mesh when the object is destroyed. Essentially this new mesh will act as the dead geometry.

Name this object `remain`. It will be used as the permanent post-destruction submodel, which replaces Main when the object's state changes from alive to dead.

[ If no node is named **remain** then all pieces other than **main** will be destruction pieces.]

This model should consist of different and realistically broken pieces that will be spawned as the destroyable object is destroyed.

Each one of these pieces needs to have a physical proxy.



Before exporting the object, place all objects at origin, so that they are intersecting each other.

The orientation in world coordinates should be 0/0/0 for all the pieces.

Turn OFF **Merge All Nodes** and click **export**.



The CGF must be exported with the **Merge Multiple Nodes** checkbox deselected, to produce multi-piece geometry.

Object can now be placed in Sandbox as a destroyable entity.

By default, the pieces spawn in the position in which they are placed in the CGF, relative to the original model.

How it works...

A destroyable object can be in two distinct states, an "Alive" state or a "Dead" state.

In the "Alive" state, the object acts precisely like you'd expect a normal physical entity to react. It can be set up to be a rigid body or a static physical entity.

After taking more damage than the specified "health", it will go into the Dead state. When going into the Dead state, as a destroyable object entity it can optionally generate a physical explosion and apply area damage on the surrounding entities. It can spawn a particle effect, and finally replace the original geometry of the entity with either destroyed geometry and/or pre-broken pieces of the original geometry. When the object breaks due to a hit from a projectile, this hit impulse is applied to the pieces in addition to any explosion impulse that occurs on the entity.

There's more...

As there can be many different types of breakable assets used within the CryENGINE you will likely want to know more about the different types of destructible and physical reactions you can create for objects.

Two-dimensional breakable assets

Using 2D breakable objects is a useful technique for level objects like glass, ice, wood, or walls. The technique works very well with thin and mostly flat mesh objects. This is controlled through the surface type of an object set in its .MTL file.

Some surface types have been specifically set up for two-dimensional breakage.

Some rules for these objects are that the object must be seven times thinner, in the direction of breaking, more than the length of the other two axes. Each triangle on the object cannot exceed 15degrees deflection between them, so keeping these meshes flat is a good idea.

Jointed breakables

Breakable objects are also sometimes referred to as jointed breakables. These are structures that are built of separate meshes being held together by virtual joints rather than having sub-models spawn. On destruction all the pieces are already rendered.

The parts can be individually disjointed by applying physical force bigger than the joint limits by using things like a gun, or by player interactions such as explosions, and so on.

Some good examples of breakable objects are road signs, wooden fences, or a wooden shack. All these assets are made of parts that are assembled, like in the real world.

Breakable objects are generally placed as brushes or geometry entities. However, you can also place them as BreakableObjects Entity.

For furniture or anything that doesn't require a base attached to the ground that remains after it's broken, it is better to use a rigidbodyEx or a basicEntity. If you need the remaining part to stay static, then it is better to use a breakable object, and specify the mass of the part as 0. If this isn't done, the engine will determine – at random – which part 'remains' as the static object.

User defined properties

Some parameters can be added to individual objects in a destroyable object asset:

- ▶ `generic = count`: This causes the piece to be spawned multiple times in random locations, throughout the original model. The count specifies how many times it is spawned. There can be multiple generic pieces.
- ▶ `sizevar = var`: For generic pieces, this randomizes the size of each piece, by a scale of 1-var to 1+var.
- ▶ `entity`: If this is set, the piece is spawned as a persistent entity. Otherwise, it is spawned as a particle.
- ▶ `density {{density or =mass}} = mass`: This overrides either the density or the mass of the piece. Otherwise, it uses the same density as the whole object.

See also

- ▶ Having learned how to use physics and breakables, go to the *Chapter 11, Fun Physics* to use your destructible objects in unique ways

Using advanced material editor parameters to create animation

In many cases, it is required to animate textures scrolling or oscillating in their uv space. This can be used efficiently for things like computer screens or TV screens that have animated noise.

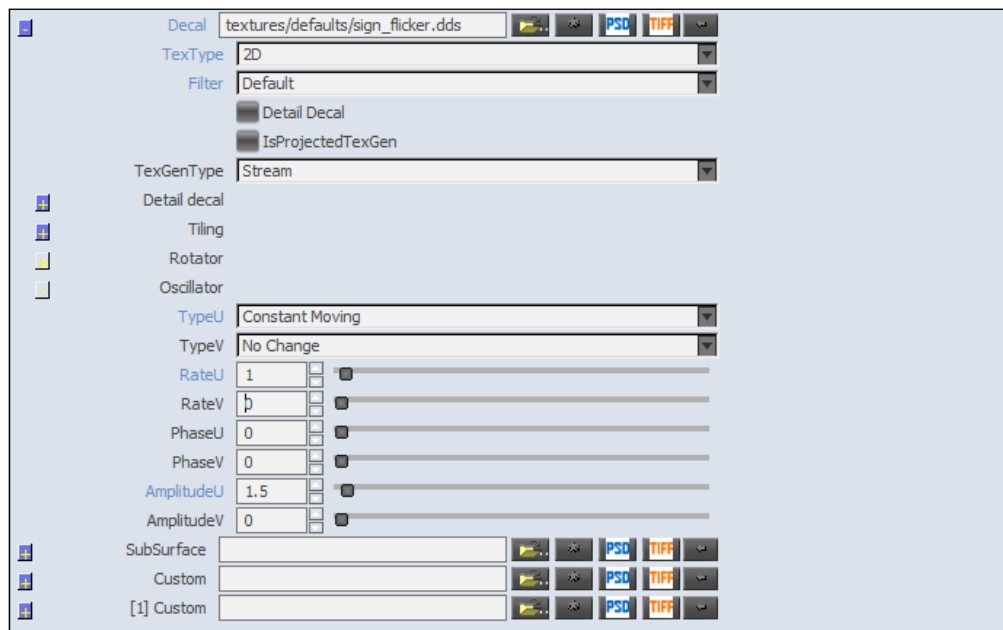
Getting ready

The Sandbox must be open and you must have applied a basic material to a solid.

How to do it...

Let's start using some of the advanced material parameters available to us:

1. Clicking a **+** sign beside any of the texture slots will open some advanced parameters for the texture.



2. In this recipe, let's set an interference texture shipped with the SDK to the Decal slot of an object.
3. Next, set the **TypeU** parameter to constant moving.

4. Now, set the rate and amplitude to **1** and **1.5** respectively.
5. Set the glow to **1** so that you can observe this particular texture scrolling through the uv space of the object.

You can now make all kinds of variations using these parameters and can experiment using animation on the diffuse slot and other textures!

How it works...

In this example, we have created a simple interference-type effect that can easily be applied to any screens or electronics in your level.

This technique is not only limited to being used for interference textures but it's also used often with any flowing type of objects. A good example is to use this on small rivers or lava.

There's more...

There are other techniques that can be used when adding variations and advanced material parameters. Some of these are explained next.

Animated textures

In order to animate textures in Sandbox, the texture name in the Editor must be set up according to specific naming conventions.

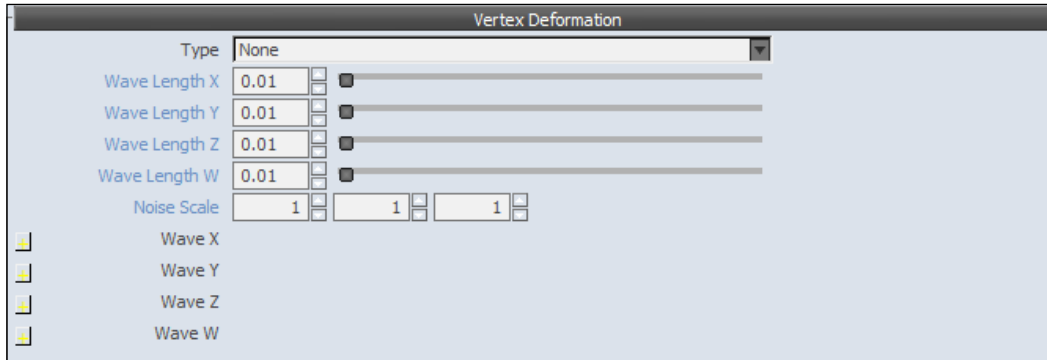
The technique is used to switch sequentially between a certain amount of textures at a particular rate:

- ▶ `prefix##ns_nesuffix(time)`.
- ▶ `prefix` – is the Start of the texture name.
- ▶ `##` – is the number of digits in the animated sequence. Putting two `#` symbols allows for sequences of over 10 frames.
- ▶ `ns_ne` – are the first and last numbers of animated sequence images.
- ▶ `suffix` – optional end of texture name.
- ▶ `Time` – optional time of single frame in seconds (default value is 0.05 seconds = 50 ms).

An example of an animated format is `bubbles##00_12simulation(0.05).dds`.

This means the texture sequence with the names: `bubbles00simulation.dds`, `bubbles01simulation.dds`, `bubbles02simulation.dds` through `bubbles12simulation.dds` is animated with the speed of 50 ms per frame.

Vertex deformation



The CryENGINE material system allows us to animate vertex deformation from some shader technique.

There are many settings available here that can be experimented with. Some specific examples of using vertex deformation is on a piece of cloth blowing in the wind or other soft objects.

See also

- ▶ Go to the *Creating basic shapes with solids* recipe in *Chapter 3, Basic Level Layout* to learn to manipulate solids
- ▶ Go to *Chapter 4, Environment Creation* to learn about the creation of a basic material

Creating new material effects

Material effects in CryENGINE are defined as the way a surface material reacts to other materials. For example, a metal material will react to bullet impacts differently (that is, by generating sparks) than a grass material does (that is, by generating dirt or dust). Because hardcoding these effects in C++ would require a huge amount of maintenance, these effects are exposed through a number of small asset files.

Getting ready

It is important that the files involved in this recipe are explained.

The `SurfaceTypes.xml`, defined in `<CryENGINE_root>/<game_folder>/Libs/MaterialEffects/SurfaceTypes.xml`, defines the physical properties of all the available material types. It can be edited with any text editor. The `MaterialEffects.xml`, which usually resides in `<CryENGINE_root>/<game_folder>/Libs/MaterialEffects/MaterialEffects.xml`, defines the interaction of two materials as it defines what effect to generate on an interactive event between two surface types. For example, it defines that when a bullet collides with the soil surface, a dirt particle effect is spawned. This file must be read and edited with Microsoft Excel. The Effect libraries, found in `<CryENGINE_root>/<game_folder>/Libs/MaterialEffects/FXLibs`, contain the associated effects.

How to do it...

Let's create a material effect.

Material effect events are those events that occur on interactions between two surface types, as defined in `MaterialEffects.xml`.

In this example, we will add a flow graph that is to be played as the event:

1. The first thing that you should do to make your own FG effect is to create a new `.xml` in `Game\Libs\MaterialEffects\FXLibs`.
2. This file should be saved as `newfx.xml` and its content should be as follows:

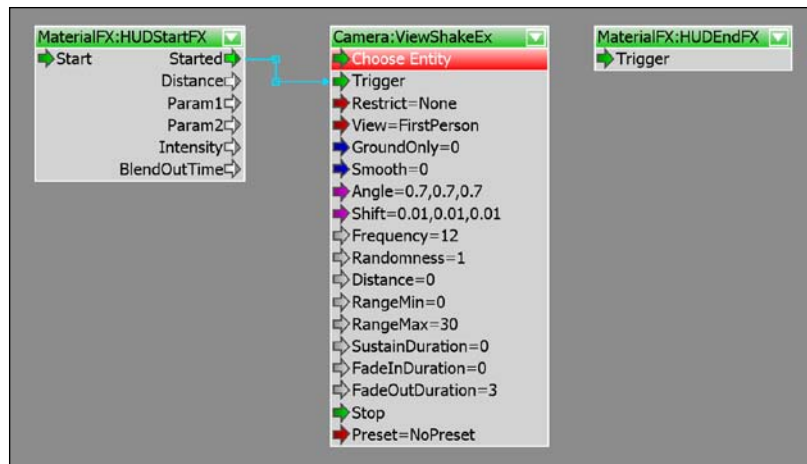
```
<FXLib>
<Effect name="hit_mat_wood_custom" delay="0.05">
  <FlowGraph name="hit_effects_fg" maxdist="10"/>
</Effect>
```

In this example, I've set this event to fire a flowgraph called "hit_effects_fg" within a maximum distance of 10 meters from the event's location.


Now that we have a new entry to access, we need to add it to the `MaterialEffects.xml`. Open the `MaterialEffects.xml` in Excel.

8
mat_wood
collisions:metal_wood
collisions:metal_wood
collisions:metal_rayproxy_wood
collisions:metal_shell_wood
collisions:glass_wood
collisions:glass_wood
collisions:wood_wood
collisions:wood_wood
collisions:wood_wood
collisions:snow_default
collisions:ice_default
collisions:water_default
collisions:rock_wood
collisions:rock_wood
collisions:rock_wood
collisions:rock_wood
collisions:soil_default
collisions:grass_default
collisions:vegetation_default
collisions:fabric_default
collisions:rubber_default
collisions:rubber_default
collisions:flesh_default
collisions:flesh_default
collisions:flesh_default
collisions:flesh_default
collisions:flesh_default
collisions:flesh_default
collisions:flesh_default
collisions:flesh_default
collisions:flesh_default
collisions:flesh_default
newFX:hit_mat_wood_custom
collisions:metal_wood
bulletimpacts:hit_mat_default
newFX:hit_mat_wood_custom
bulletimpacts:grenade_hit_mat_de
bulletimpacts:hit_mat_wood
bulletimpacts:melee_hit_mat_wood
tornado:leaves
collisions:grenade_default
bulletimpacts:grenade_hit_mat_de
bulletimpacts:grenade_hit_mat_wa
footsteps:wood_solid
player_foley:bf_woodsolid
vehicles:truck_grass
vehicles:truck_grass
collisions:medium_weapon_wood

3. Enter the new MFX event. The previous example enters the event into the "bullet" row and the "mat_wood" column. You can see the way the material event is read by entering the new event as the `libraryname:eventname`, so mine is `newFX:hit_mat_wood_custom`.
4. Create a flow graph.



The previous example shakes the camera for the player.

[ The two nodes that are critical to be in the flow graph are **MaterialFX: HudStartFX** and **MaterialFX:HudendFX**. If those two nodes are not there, it simply won't work. The endFX node does not need to be attached to anything if not required, but still must be present.]

5. Once you have created the flow graph, go to **File | Save As** in the flow graph editor and save it with the name you registered in the event earlier in the `<FlowGraph name="hit_effects_fg" maxdist="10"/>` line of the event. For this example, it is `hit_effects_fg`.
6. Now that it's all saved and the events are being called when the bullet impacts the wood, start up sandbox and go shoot some wood and you'll see that the event triggers and the flow graph is activated.

How it works...

The cross-reference between two surface types defines the type of event that will happen on interaction. Surface types that are only called by code are required to only have rows and must be added at the bottom, below the surface types defined in `SurfaceTypes.xml`. (The exception is when it is needed to be used on materials through the Material Editor).

There's more...

There are many parameters that can be added and tweaked to the Material Effects system, such as bullet pierceability through materials and creating all new surface types.

Creating new surface types

A surface type is defined by the XML element `SurfaceType`. Its attribute `name` is the one that can be selected in the Sandbox Editor. Surface types defined here will appear as options in the drop-down list of the Material Editor. The other attribute "type" is optional. It can be used in other processes by game code. The physical parameters are defined by the "Physics" element.

New surface types have to be added both as a row and as a column, and have to be kept in the same order.

Physics block parameters

With the existence of surface effects, we can apply some advanced characteristics to materials in the engine.

Some of the parameters that are added to the physics block are:

- ▶ Pierceability is an integer from 0-15, with 0 being the least pierceable. For each ray, a hit is pierceable if ray's pierceability is less than the material's pierceability
- ▶ Friction defines a material's friction, friction is the average of contacting materials. A recommended default is 0.8. Internally, it's clamped to 0.1 if it's below that

Ammo surface types

The surface type for ammunition is the attribute "name" of the XML element "ammo" in an ammo file. Its value is required to be the same as that inserted in the MFX table.

See also

- ▶ Go to *Chapter 9, Game Logic* to learn more about Game Logic and FlowGraph scripting.

Creating image-based lighting

The concept of using image-based lighting brings many advantages but must be used carefully. As such, it is beneficial to know about its properties and combine different methods where useful.

IBL allows CryENGINE 3 to render very complex lighting situations; usually, this is done with an infinitely distant environment map.

Positive points:

- ▶ High quality
- ▶ Fast for many lights and even a complex environment is reflected
- ▶ Energy preserving specular power

Negative points:

- ▶ Works only well for local positions with distant emitters and reflection content
- ▶ Static content only

There are two distinct ways to generating an environment map or cube map from CryENGINE.

In this recipe, we will use the entity environment probe, which is the most convenient method.

Getting ready

Open a level in Sandbox.

How to do it...

Let's set up our own image-based lighting.

1. Drag-and-drop the entity `EnvironmentProbe` from the MISC section of the RollupBar.
2. Click on the entity property **preview_cubemap** to preview the cubemap.
As we haven't yet generated a cube map, it will be completely black.
3. Next, click the generate cube map button at the top of the entity.
The Engine will then automatically bake the cubemap into a texture that gets stored in `textures\cubemaps\levelname` where level name is the name of the level that is opened when the cubemap is generated.
4. You will also notice that the preview sphere now has the cubemap applied to it.
5. Set the **env Probe** to active and adjust the radius parameter to observe the projected cube map reaction.

How it works...

Image-based lighting is a rendering technique where complex lighting is stored in an environment map, which is projected onto the scene. In simple words, a light probe or environment map is just an image on a sphere.

As this object acts as a light projecting the cubemap onto any objects within its radius, it allows the artist to create realistic environment maps based on the actual environment the object is in, rather than using a generic environment map.



Even a single global light probe helps to improve the lighting, as the classical flat ambient is replaced by something that is direction-dependent and shows some specular and material behavior.

There's more...

There are many properties further available in the environment probe as well as a second way to create cubemaps.

Generating all Cubemaps

There is another function in the environment probe entity, which is the ability to bake all cubemaps for all light probes in a particular level.

This is quite useful for updating the environment maps throughout the entire level very quickly.

Creating a CubeMap with Material Editor

To create a local cubemap with the Material Editor, follow these steps:

1. First, create and select the object that will serve as the center of the camera that will record the cubemap. In this example, the `palette_box` brush is used.
2. With the object selected, open the Material Editor and click **Get Material From Selection**.
3. With the object's material selected, click **Generate Cubemap** for the selected object.
4. Enter a name for the cubemap, and click **Save**.
5. Enter the resolution of the cubemap. **256** is the default resolution.

See also

- To get the best out of image-based lighting it is best to have a level. Go to *Chapter 3, Basic Level Layout* to learn to create your environments

7

Characters and Animation

In this chapter, we will cover:

- ▶ Creating skinned characters for the CryENGINE
- ▶ Ragdoll and physics for characters
- ▶ Creating animation for your character
- ▶ Previewing animations and characters in Sandbox
- ▶ Creating only upper body animations
- ▶ Creating locomotion animations
- ▶ Animating rigid body geometry data

Introduction

In this chapter, we will explore the creation of animated characters for the CryENGINE. We will also extensively cover the creation of different animations for a variety of asset types.

Animated characters are integral to any game as any time you want to have realistic-looking humans, aliens, or anything of that sort you need to be able to skin the asset's mesh to a skeleton. Once the mesh has been skinned then it can be exported to the engine as a `.CHR`. There is a second format, used mostly when dealing with vehicles and rigid body assets, which is called `.CGA`.

A character is a combination of geometry data that is attached to a skeletal hierarchy. Examples include a human body, a shark, an alien, a horse, or a rope. In this chapter, we will mainly be dealing with humans; however, keep in mind that most of these recipes can be applied to other types of assets.

Throughout this chapter, we will be dealing with the SDK sample files shipped with the CryENGINE3 SDK. With that said, however, there is still extensive information throughout this chapter on creating a completely new character from start to finish and thus it can be used as a reference when creating a custom character.

Creating skinned characters for the CryENGINE

The CryENGINE has a robust character and animation system.

It is however important to realize that, for the character creation process to go smoothly, you should follow the recommended workflow put forth in this first recipe for the creation and subsequent animation of skinned characters.

Getting ready

Locate the character assets that are shipped with the CryENGINE SDK.

In most cases, a character subfolder is created within your particular game's object folder.

In the SDK, the default character is located in `Game\objects\Characters\netural_male`.

When creating character assets for the CryENGINE, you will typically save them to two or three different `.Max` files. The first file will contain the main character; the second scene contains the first LOD of the character and its ragdoll physics mesh. The third file is optional, as it usually contains just the head of the character. The head is separated as it makes it easier to edit independently of the rest of the character.

Before we begin, some important things to note about the geometry used for characters for the CryENGINE are:

- ▶ Characters must be facing forward on the Y+ axis in 3ds Max
- ▶ The skinned geometry must have no transformations applied and its pivot should be set to origin, or 0,0,0

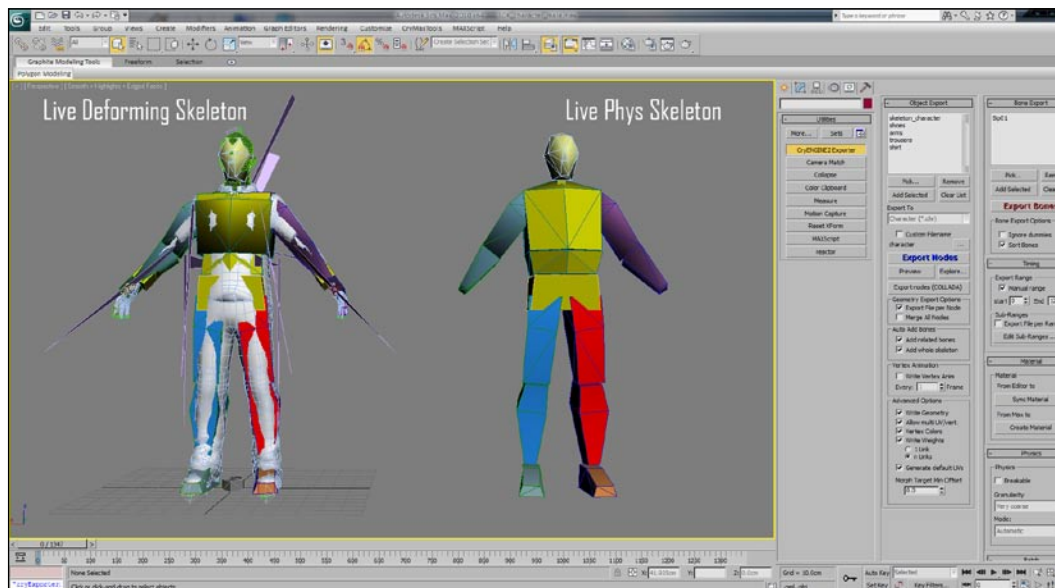
As there are many tutorials available for character modeling and texturing, we will not go into deep detail of the actual modeling and texturing process. Rather, we will learn to manipulate the SDK character and explore how we can use this to build our own character.

How to do it...

Let's use the sample skeleton to learn how to create our own character:

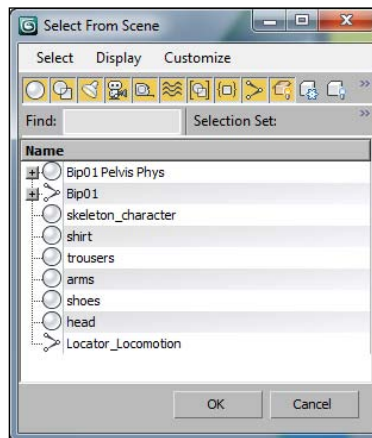
1. Locate the Max file named `SDK_character_male.max`, which is stored under `Game\objects\Characters\nneutral_male`.

You should now be viewing the neutral male character scene, as shown in the following screenshot:



2. Let's explore the nodes within the Max file:
 - ❑ The deforming skeleton is a typical starting point for all characters as it is responsible for many tasks
 - ❑ It deforms the render meshes of the character based on skinning information
 - ❑ The actual geometry of the bones in this skeleton will be used for hit detection and physics for an active or alive character
 - ❑ Materials applied to the deforming skeleton are also used for hit detection and can be used to apply physical impulses or to read where the character has been hit

3. Open the **select by name** list in 3ds Max by pressing the shortcut *H* or by clicking the select by name icon.



4. All the nodes under and including **Bip01** are what comprise the Live Deforming Skeleton.



It is strongly recommended that for human characters you retain the same hierarchy as the example scene. New bones can be added but they should not interfere with the 3ds max biped's default hierarchy as there are additional nodes used for automatic foot plant and ground alignment.

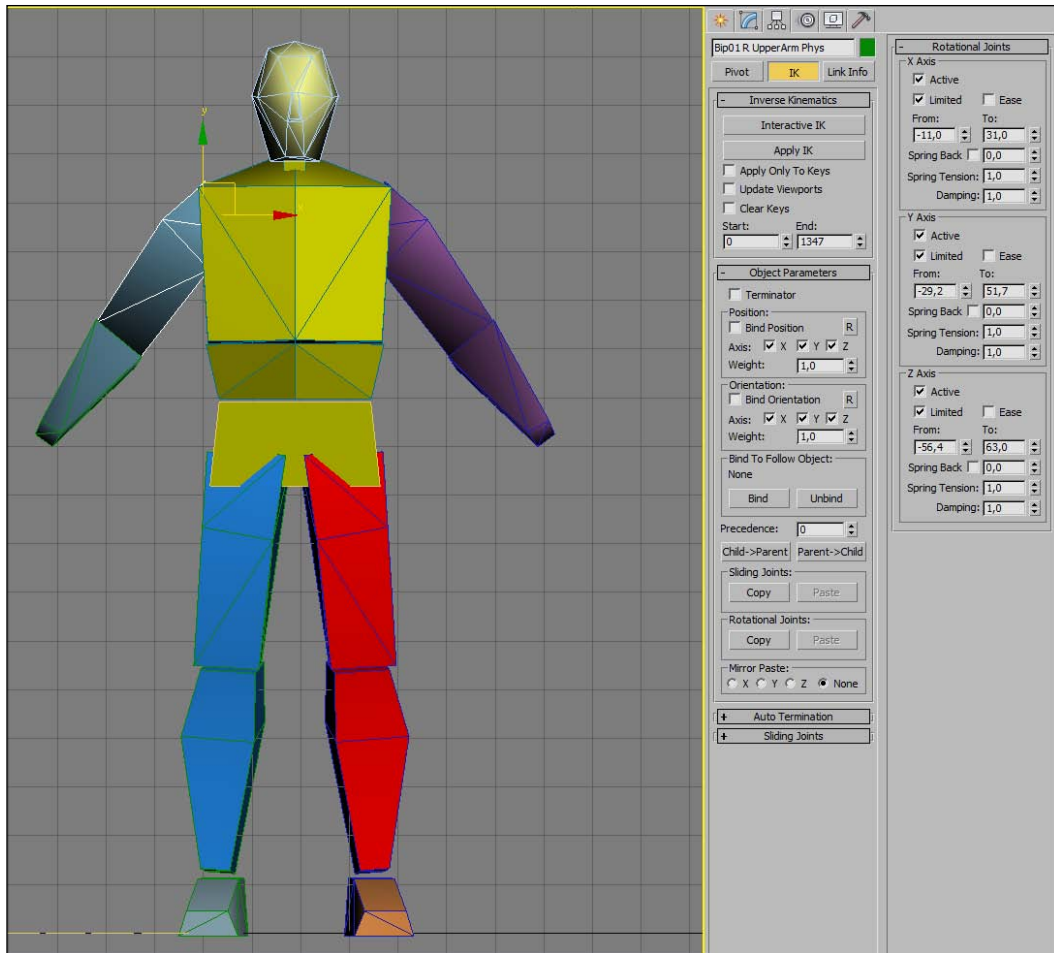
5. At this point, if you are working on a custom character, you will want to import your custom mesh into the `neutral_male.max` scene.



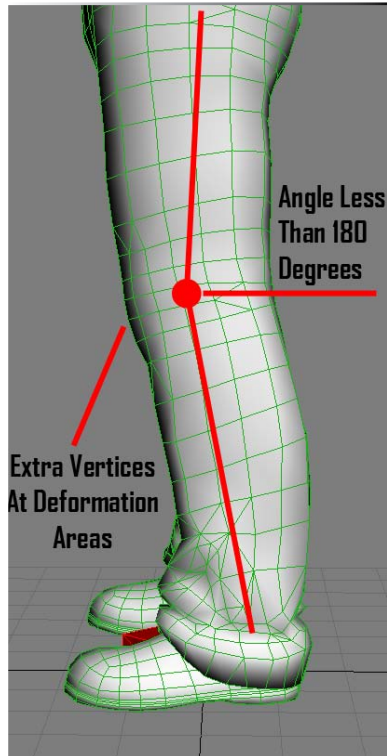
When creating your own character, you do not necessarily need to match your render mesh to the SDK skeleton. However, keep in mind that if you do not use the `neutral_male` Live Skeleton, it will mean that your custom character will not be able to use the existing animation set that ships with the SDK.

To achieve high fidelity animations, current generation games can include hundreds of individual animations, so to get a character in game quickly it's best to use the pre-built skeleton.

6. The second skeleton that must be in the scene when you create your character is the **Live Physics Skeleton**, also called the **Live Phys Skeleton**.
7. The best way to understand the Live Phys Skeleton is to think about it in terms of a set of constraints to each bone in the Live Skeleton. When a bone is present in this Phys skeleton, it signals its counterpart in the Deforming Skeleton that it should be physical in the world when the character is alive. The Phys skeleton for your character also stores physical properties for its counterpart which are stored in the phys skeleton bone's IK properties, as seen in the following screenshot:



Now that we understand the function of both the skeletons in the main character max file, we will discuss the setup of different portions of the character. As you create the mesh, you will find it significantly easier to skin and animate your character if it properly matches the existing skeleton. Another important consideration is the density of vertices at key areas, such as the knees and elbows. Spending a few extra vertices in these locations will greatly increase the quality of the deformation when animated.



8. Always make sure that, whether you use the SDK skeleton or your own custom skeleton, the model matches the joints in the skeleton.
9. Having created geometry for the character, you will have to skin this geometry to the skeleton for animation.



The Physique and CrySkin modifiers can be used in the place of the skin modifier.

10. This can be done by applying the 3ds max modifier skin to your geometry and by adding the Live Skeleton bones that should affect that character into the skin modifier.

11. When applying the skin modifier, ensure that your skeleton is in the bind or neutral pose.
12. Before exporting your character mesh, there is a technique that can be used to make it very easy to modify certain parts of the character in the engine directly.
13. This can be achieved by exporting a `skeleton_character`, which is simply a single triangle usually hidden in the pelvis or set to a no draw material skinned to the entire live skeleton. This will allow you to export a blank skeleton onto which you can add your attachments using the Character Editor.
14. If not already done, add the `skeleton_character` object to the export nodes list in the CryENGINE exporter.
15. You will notice that when adding a skinned object to the export nodes list, it will, by default, automatically add the required skeleton to the export.

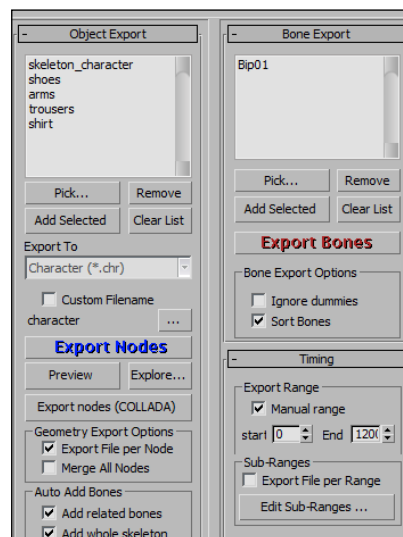


The exporter automatically adds all children bones of the listed skeleton node.

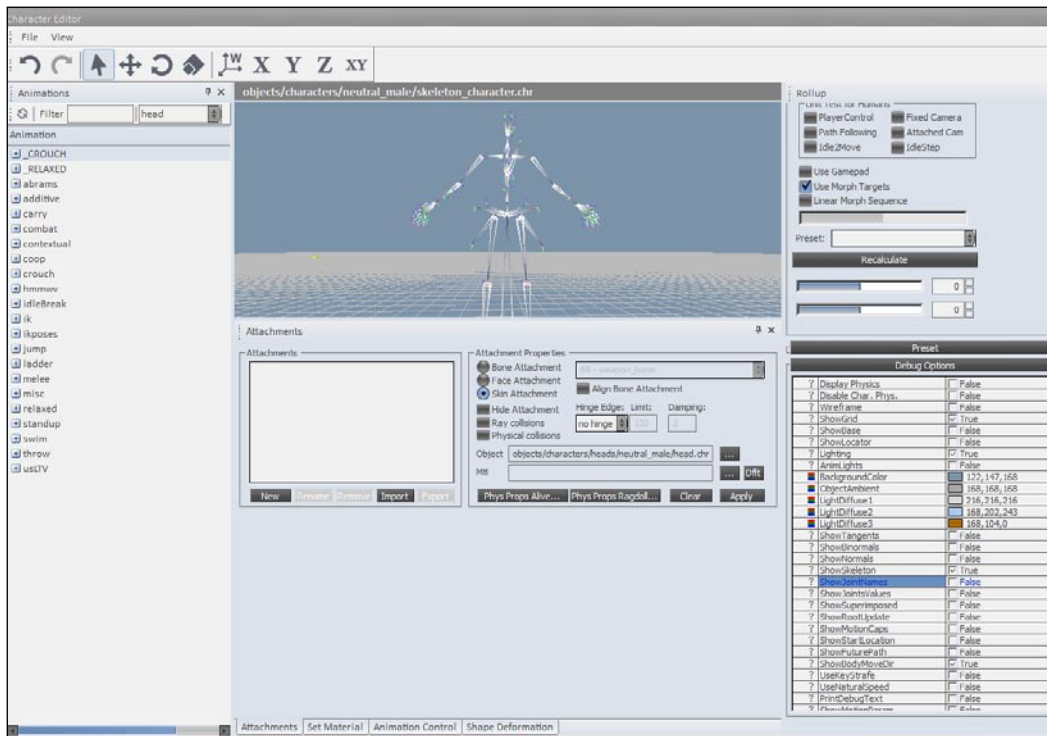
16. As we will be exporting this file as a character, the format we are required to use is the `.chr` format.

When managing multiple character meshes in a single scene, it is important to use the export file per node parameter in the export settings. This will use the name of the object in 3ds max as the name of the `.chr` created. The added benefit is a one-click export of any number of character meshes.

17. Let's now add the remaining characters in this scene to the export list. When you are done, the CryENGINE exporter should appear as shown in the following screenshot:



18. Once you have added the remaining characters to the scene, you can now export the render meshes and the skeleton!
19. To do this, click the **export nodes** button.
(Good workflow dictates that we test our export in game.)
20. Open Sandbox. Click open **View | Character Editor**.
21. Using the **File | Open** dialog in the character editor, you can now open the exported .chr files.
22. As we want to assemble our character into a single manageable file, we can now create a .cdf (Character Definition File) using the character editor.
23. To do this, click on **File | Open** and open the `skeleton_character.chr` file.
You can preview the skeleton by adjusting the debug options at the bottom of the character editor.



24. Let's now begin to add the attachments that will make up our character:
 - In the **attachment properties** select **skin attachment**.
 - Then under the attachments window click **new**.
 - Name the new attachment `shirt`.

- ❑ Select the **browse** button beside the object string and browse to the `shirt.chr` we exported earlier.
- ❑ Leave the **material** field blank for now as it takes the default material applied to the attachment model.
- ❑ To view the attachments on the character click **apply**.
- ❑ You will see that the shirt is now attached to the skeleton and if an animation is played, the shirt reacts to the parent skeleton appropriately.

25. Add the remaining pieces of the character we exported earlier.

26. Now that we have assembled this character, we need to save it for later use in engine.

- ❑ To do this, navigate to **File** and click **Save As** and save the file as `.cdf`. You can name it as you see fit for this example; I will save it as `cookbook_character.cdf`.

This can now be used later as the player model, AI friendly, enemy characters, and animated cinematic characters!

How it works...

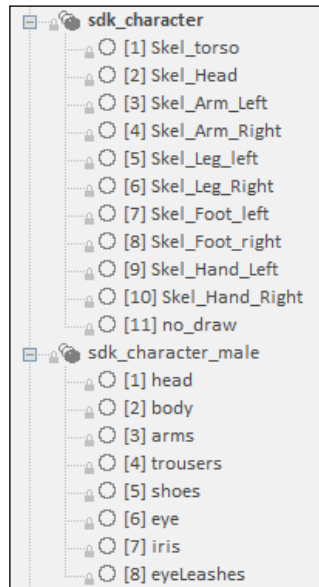
Characters allow us to use complex animation and locomotion on characters and thus be able to create working and realistic characters imperative when creating a game. It can be used as the player's representative model in game, in cut scenes, and finally as artificial intelligence.

The attachment system is one of the principle reasons behind the Character Editor. Current generation games can have upto 20 to 30 different character-models and thus, using attachments is one way to avoid repeatedly displaying the same character over and over again to the player. The attachments system also allows designers and artists to give the character a unique look in a very intuitive and easy way.

Materials and characters

When working with an example character, you will have noticed that the bones of the skeleton were colored and thus had a material applied from within 3ds.

When creating materials for characters, this is an option available to you as it can be used later on via code or other means to identify areas where characters have been hit. Sometimes, characters are simple enough that the render materials can be added in addition to the bone materials in one larger multi subobject material, or they can be separated as in the following example:



You can see that the `sdk_character.mtl` contains sub-materials for each physical bone as well as a `no_draw` material for the simple triangle used in the `skeleton_character`.

LOD (Level of Detail)

A character **LOD (Level of Detail)** is in geometry with a lower polygonal resolution that will be faded automatically in-between based on the character's distance from the camera in the game engine.

To create LODs, you must copy the original character mesh and then reduce its triangle count, which can be done manually, or for the sake of speed you can use automated modifiers in max such as Multires.

Once the polygon count has been lowered, the fastest way to skin the LOD (similar to the main character) is to use the skin wrap modifier in 3ds max and use the original mesh and skinned mesh as the target. You can then convert this skin wrap modifier to a regular skin modifier and then export the newly created LOD.

LOD objects must follow the `_LODn` naming convention where `n` is the number of the LOD.

Bone Attachments

When using the character editor in sandbox, you can add other types of attachments to characters other than skin attachments. One such attachment is the bone attachment. Adding bone attachments is a way to have any model, be it a .cgf, .cga, and other .chrs, constrained to a single bone within your character. This can be useful for rigid body attachments such as weapons and props such as sunglasses or helmets. When applying bone attachments, you may sometimes have to align the bone attachment. When set, this aligns the pivot of the geometry with the bone. In other words, the rotation of the joint and the rotation of the attachment will remain identical throughout animations.

See also

- ▶ Use your character in interesting ways by going to the *Hangman on a rope* recipe mentioned in *Chapter 11, Fun Physics*
- ▶ Carry on to the next section to learn how to set up the ragdoll reaction for characters

Ragdoll and physics for characters

In CryENGINE3, characters have realistic physics applied to them. When a character is reacting only physically to the world and not playing a specific animation, we typically call them ragdolls. Ragdolls are usually used to make every death of an AI character look unique and exciting; this can be very rewarding and fun for the player!

Working with and editing ragdolls can be a lot of fun. This recipe will take you through the setup and theory of using ragdolls in CryENGINE3.

Getting ready

As mentioned previously, characters are stored in multiple Max files.

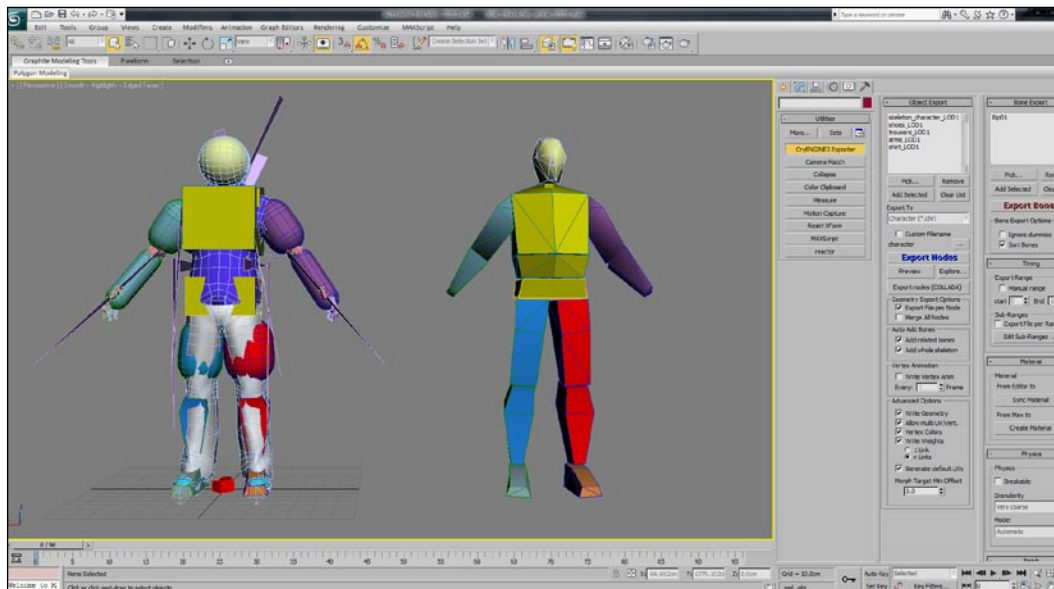
In the SDK, the default character is located in `Game\objects\Characters\nneutral_male`.

In this case, we are interested in the `SDK_character_male_LOD1.max` file.

How to do it...

Let's begin by getting accustomed with the sample ragdoll file:

1. Open the SDK_character_male_LOD1.max file.



You will notice similarities with the main character file mentioned in the previous recipes. The deforming skeleton in this scene is used as the dead physical skeleton of the character. Because this skeleton will react physically with many objects in the world, it is important that it be made from simple primitives, such as capsules, boxes, or spheres.

The second skeleton in the scene, the Phys skeleton, acts identically to the main character Phys skeleton, except that it acts as the set of switches and constraints for only the ragdoll's deforming skeleton. If you are creating a custom character, creating this _LOD1 file is not complicated.

A very good technique to create the geometry for the ragdoll's deforming skeleton is to align some primitives created in max to the existing live skeleton. Using the **editpoly** modifier on the bones, you can then attach the new primitive geometry to your bone and remove the old bone geometry.

2. Having set up the geometry for physics and this ragdoll, we can now save the file with the _LOD1 suffix.

3. Next, you can create the first LOD of your render mesh character attachments manually or by using a more automated process such as **multires**.
4. Ensure that the objects in max also have the `_LOD1` suffix as we will be using export file per node when we export these characters.
5. Once you are ready, click the **export nodes** button. There are really two different ways to test the ragdoll physics of a character in the CryENGINE.
6. Open Sandbox and open any level.
7. Now, place an AI entity into the world and ensure that its model parameter is set to either your custom model or, if you are using the SDK skeleton, ensure it is set to `Objects/Characters/neutral_male/sdk_character_male_v2.cdf`.
8. Next, as we are testing the physics enter the following console variable in the console:
`p_draw_helpers = 1`
9. Now you can see the live deforming skeleton rendered as a pink geometry:



10. You should now enter into game mode (*control G*) and shoot the character until his health reaches 0.

11. Once the character's health runs out, the entity will switch to ragdoll.
12. Another way you can quickly kill the AI entity is to right-click the entity and to select the kill event under select events.
13. You will see that, when the character is killed, all the bones react according to the IK settings in the Phys skeleton of the `_LOD1.max` file.



14. The final and – in my opinion – most effective way to test the ragdoll of a particular character is to use a dead body entity.
15. You can find this entity under the **Entity | Physics** tab.
16. Drag-and-drop this entity into the level and set the model path to your custom character or to the sample character. The model path is as follows: `Objects/Characters/neutral_male/sdk_character_male_v2.cdf`.
17. By default, resting is set to `true` on the dead body; ensure that this is set to `false` to have the ragdoll simulate.
18. You can enable the simulation by pressing the **AI/Physics** button or by entering game mode.

How it works...

When a character is killed, the physics system takes over the character as we do not need to play a certain death animation with an effective physics system. The character's skeleton then takes the `_LOD1` live deforming skeleton's geometry and simulates the bones according to the IK settings within `_LOD1 Phys` skeleton.

There's more...

You may want to know how to adjust the physical characteristics of the ragdoll further or how to save the physical state, or pose, of a ragdoll character.

IK limits

The IK limits applied to the Phys skeleton are the key to a good looking ragdoll simulation! These limits are applied as IK rotational limits from the bone in the dead body Phys skeleton.

Editing this IK information can take a good deal of testing. To adjust the IK limits for a bone in the phys skeleton, you can select it in 3ds max and go to the **hierarchy** tab in the rollout bar. Under the IK setting, you will be able to access all the rotational constraints for the Phys skeleton.



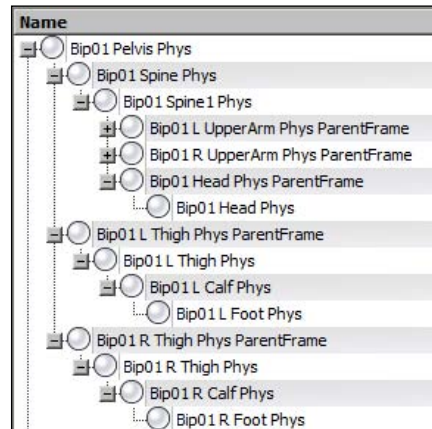
IK Y angle limits should lie in $(-90^\circ, 90^\circ)$ range.

ParentFrames

ParentFrames is a technique that can be used to allow joints to rotate beyond the $(-90^\circ, 90^\circ)$ range. To create a parent frame and an additional parent object, the bone must be created.

If the joint needs to rotate beyond the $(-90^\circ, 90^\circ)$ IK Y limit, an additional parent node for the bone can be created, and IK rotational limits will be relative to it. The objects must have the naming convention + 'Phys ParentFrame'.

An example setup can be seen in the SDK sample character .max file:



When the angle of a particular bone varies greatly from its immediate child, it is recommended to use a ParentFrame object.

Dead body entity settings

Dead body entities can be used in levels as well as for testing character physics. Typically, workflow is to create the entity with the resting set to false and enable AI/Physics to simulate it in the editor. Then, the physical state can be saved by using the **Save State** button in the entity parameters.



See also

- ▶ A ragdoll is impossible without having previously created a character. Refer to the *Creating skinned characters for the CryENGINE* recipe in this chapter
- ▶ To learn to create animation for your character, refer to the following *Creating animation for your character* recipe

Creating animation for your character

Characters wouldn't be very interesting without animations! In this recipe, we will cover the basic task of animating a character and exporting that animation to be played later in Sandbox.

Getting ready

There are four important steps involved in preparing and animating your character from CryENGINE 3:

- ▶ Creating a character and export a .chr file
- ▶ Entering the character's information into the .cba file
- ▶ Animating the character and export a .caf file
- ▶ Adding the exported .caf file to the character's .chrparams file

Use the SDK sample asset character to start this recipe, as the character creation process has already been discussed in previous sections.

How to do it...

First, let's open the character in the character editor:

1. Open the character editor and open the `Objects/Characters/neutral_male/sdk_character_male_v2.cdf`. It is important to verify that your character is working in the engine before we can begin with animation.
2. We will now open the `animations.cba` file, which can be opened in any text editor. You will find this file in the `game/animations` folder.
3. In the CryENGINE SDK, the first entry is the sample character. You will notice that comments already exist to explain each of the parameters.
4. If you are creating a custom character, then the best practice is to copy the definition for the sample character and set the Model File path, Animation path, and Database path.

```
<AnimationDefinition>
  <!--the reference model-->
  <Model File="../../../Objects/Characters/neutral_male/skeleton_
character.chr"/>
  <!--the path with ALL animation we can use on this model-->
  <Animation Path="human_male"/>
  <!--for all animations use compression level 1-->
  <COMPRESSION value="2"/>
  <RotEpsilon value="0.0000001" />
  <PosEpsilon value="0.0000001" />
  <Database Path="human_male/human_male.dba"/>
  <!--for all animations need to detect footplants-->
  <FOOTPLANTS value="NO"/>
  <!--we apply the Locomotion_Locator modification just on BIP-
files. Aliens, vehicles and weapons don't need it. -->
  <LOCOMOTION_LOCATOR value="YES"/>
```

```
<!--we apply different modifications to the weapons . Human
and Aliens don't need it. -->
<!--a list of animation that need special handling-->

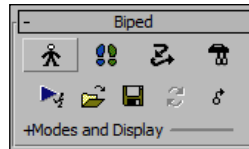
<SpecialAnimsList>

    <!-- AIMPOSES COMMON -->
    <Animation APath="Aim" footplants="NO" compression="0"
autocompression="0" SkipSaveToDatabase="1" />
    <Animation APath="vehicle" footplants="NO"/>

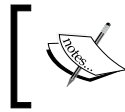
</SpecialAnimsList>
</AnimationDefinition>
```

Having verified that this character is already present in the .cba, we can begin animating. Should you be making a custom character, ensure your character is registered in the .cba before animation begins. Let's create a simple arm waving animation:

1. To animate the character open the sample character Max file.
2. Make sure that **biped** is not in figure mode as you will not be able to keyframe any bones.

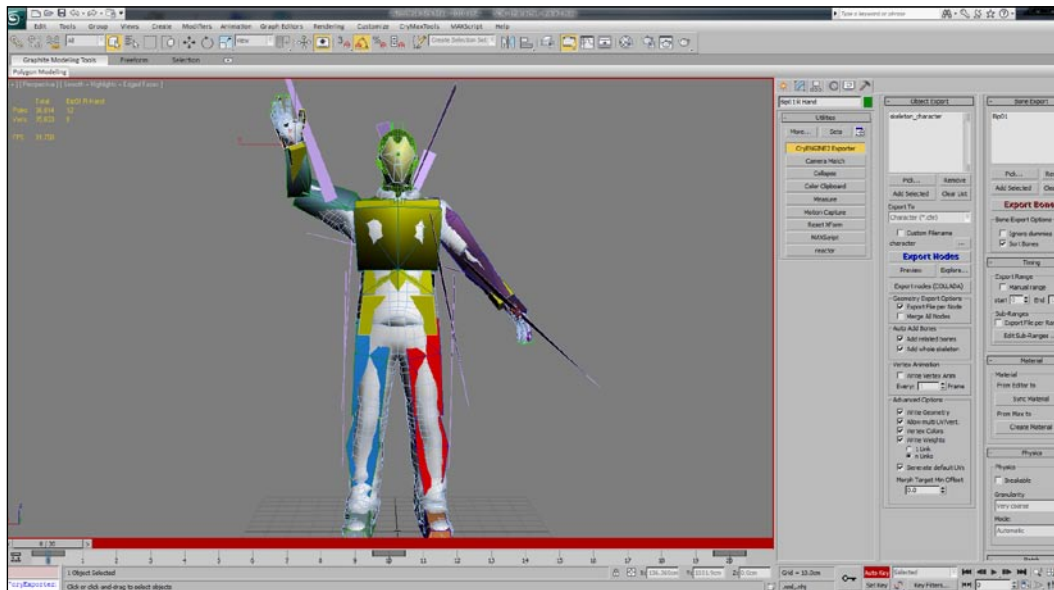


3. As this is a 3ds max biped, you can now apply motion capture data to the live deforming skeleton.
4. Set the animation mode in 3ds to **Auto Key** and move the right-hand bone into a wave position. You will notice that once you move the bone, a keyframe is automatically created.
5. You may have to rotate the upper and lower arms to get a more realistic pose.
6. Once you are happy with the position move the time slider to the tenth frame and set a different pose.
7. Finally, as we want this animation to loop, we must set the last keyframe of our animation to the same position as the first keyframe.
8. This can be done easily by selecting all the bones you want to animate by using *shift* + click on the keyframe in the timeline and dragging the copied keyframe to the final frame.
9. In this example, copy the first keyframe to the twentieth frame.
10. As we will only be exporting a single animation, we can set the time configuration in max to only show frames from 0 to 20.



The exporter will automatically export the currently displayed frames in the animation time line, if not set otherwise.

11. To export the animation, open the CryENGINE exporters in the **tools** tab and ensure that the `skeleton_character` has been added to the **export nodes** list.
12. As discussed earlier, this will automatically add the parent node of the skeleton to the export bones list.



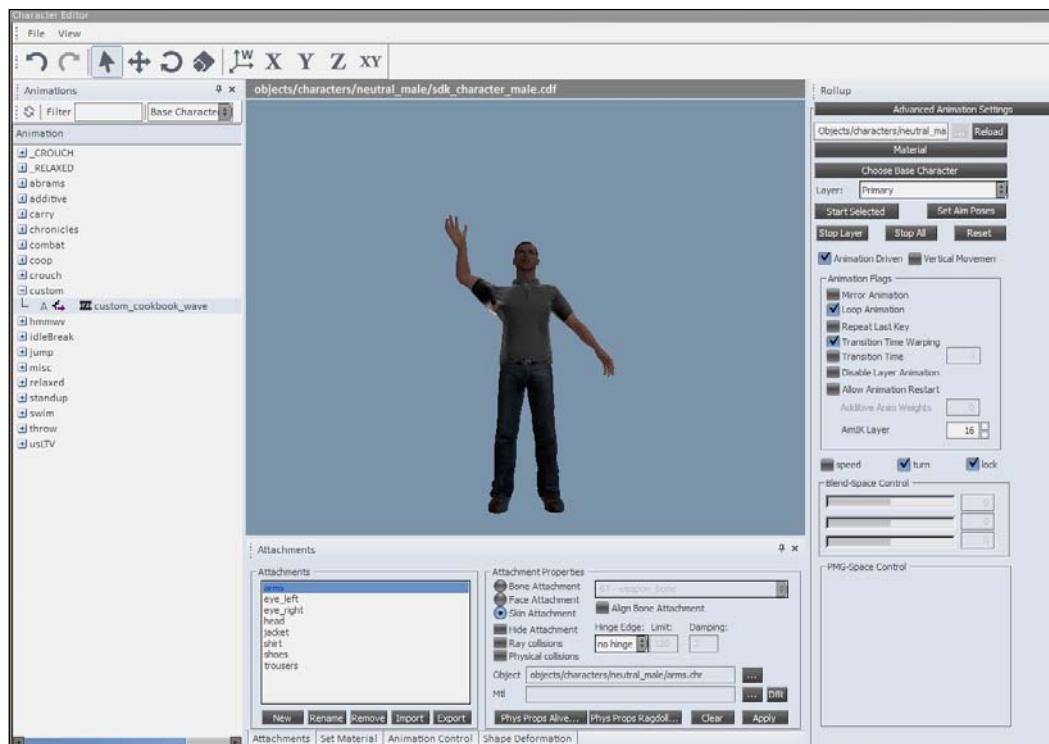
13. Clicking the **export bones** button will open the save dialog box, which allows you to save the animation as a `.caf` file.
14. Save our waving animation under the defined path in the `animations.cba` for the character. In this case, save it as `cookbook_wave.caf` under the `Animations/human_male` folder.
15. There is one final step that we must complete before being able to view our character animation in Sandbox's character editor. We must add this new animation to the characters animation list file `.chrparams`.
16. Open the `.chrparams` file in any text editor and locate the animation list. It is defined by `<AnimationList>`:

```
<AnimationList>
  <Animation name="#filepath" path="animations\human_male"/>
  <Animation name="$AnimEventDatabase" path="animations\human_
male\human_male.animevents"/>
```

```
<Comment value="$Include = objects\characters\human\
GlobalHuman.chrparams"/>
Insert a new entry for our animation below the $Include line as
below
<AnimationList>
  <Animation name="#filepath" path="animations\human_male"/>
  <Animation name="$AnimEventDatabase" path="animations\human_
male\human_male.animevents"/>
  <Comment value="$Include = objects\characters\human\
GlobalHuman.chrparams"/>

  <Animation name="Custom_cookbook_wave" path="cookbook_wave.caf"
```

17. Having now added an entry in the .chrparams file, open Sandbox and then open the character editor.
18. Using **File | open**, open the character Objects/Characters/neutral_male/sdk_character_male_v2.cdf.
19. And notice in the left-hand side of the character editor that you can now browse into the newly created custom folder where you will see the cookbook_wave animation.
20. Click on it to preview the animation.



21. You have now created and exported your animation to the CryENGINE!

How it works...

When a character is loaded by the engine, it checks for a `.chrparams` file that defines a list of valid animations for the particular character. These animations then become accessible through code, cut scenes, and the character editor to be played back.

In this example, we dealt with `.caf` animation data that is created by exporting the animation range from 3ds. Upon export the Resource Compiler is invoked and a certain amount of compression and optimization is applied to the animation to make it ready for real time playback.

There's more...

You may want more information on the compression of animations or how to expedite the character animation process by wildcarding animation names within the `.chrparams` file.

Changing the animations compression

You may have animations that are jerky or jittering. This can be caused by the original animation asset and in some cases, it can be caused by aggressive compression applied to the `animation.cba`.

The value to control compression is seen in a character's definition in the `.cba` as the line:
`<COMPRESSION value="2"/>`.

Change this value to a lower number to compress the animation less aggressively.

.chrparams file Wildcard Mapping

Wildcarding the animation names within the `.chrparams` file can be carried out as follows:

- ▶ Assigning each animation definition in the `.chrparams` file, line-by-line, can take an unacceptable amount of time and can lead to very big `.chrparams` files.
- ▶ Wildcard Mapping uses the asterisk (*) to represent the filename. Using the asterisk in the in-game name will replace it with the part of the actual filename that is wildcarded.

For example:

```
<Animation name="coop_*" path="coop\coop_*.caf"/>
```

- ▶ The previous line will load all animation from the folder `coop` that starts with the word `coop` and has the extension `.caf`.
- ▶ These animations will automatically be assigned an in-game name that starts with `coop_` and then follows the part of the filename that comes after `coop` (which will be removed from the name).

- ▶ A filename `kickroundhouse.caf` would be mapped to `coop_roundhouse.caf`.
- ▶ It is important to ensure that these folders are kept clean of old and unused assets, so as not to waste extra memory.
- ▶ One important advantage of using wildcards is that, once a folder is mapped then new animations can be quickly added to that folder without the need to add an animation definition in the `.chrparams` file each time.

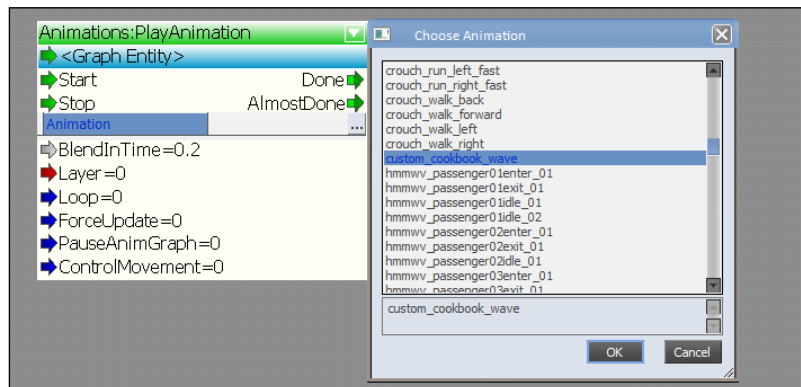
It is only necessary when the in-game name that is supposed to be assigned to the animation is substantially different than the filename of the `.caf`. Wildcard Mapping can even be used on an entire folder.

Animobject entity

A useful entity for testing animations is the animobject entity found under **Entities | physics | animobject**.

Once placed in a level, you can assign a model and then enter the string of the in-game animation name.

A good technique is to add this animobject to a flow graph and assign the animation to it using the **Animation:PlayAnimation** flow graph node.



See also

- ▶ To learn how to create a character refer to the *Creating skinned characters for the CryENGINE* recipe earlier in this chapter
- ▶ To learn more on how to preview animation within the character editor, go to the next recipe in this chapter

Previewing animations and characters for Sandbox

The character editor is a very powerful tool and is one of the numerous subsystems within sandbox. We have already used some of its basic features, and now it's time to explore a few of the advanced features and their uses as they relate to animated characters.

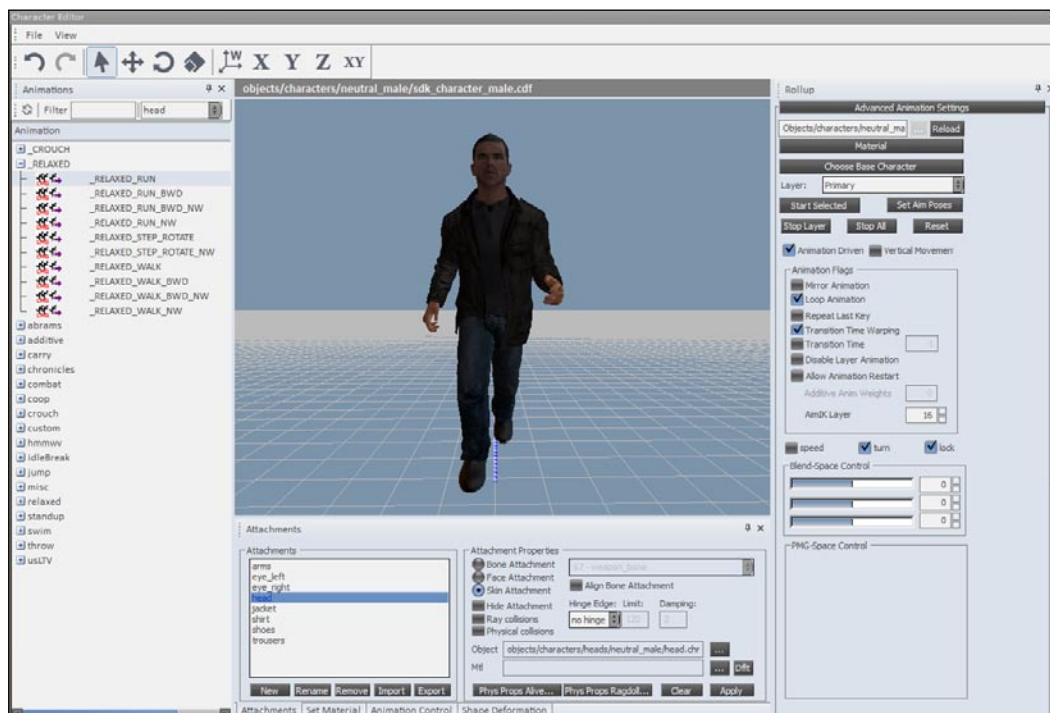
Getting ready

Open Sandbox and also the character editor. Then open the SDK sample character `Objects/Characters/neutral_male/sdk_character_male_v2.cdf`.

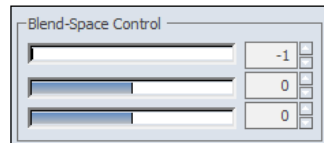
How to do it...

As we saw in the previous chapter, you can easily play animations on characters through the character editor:

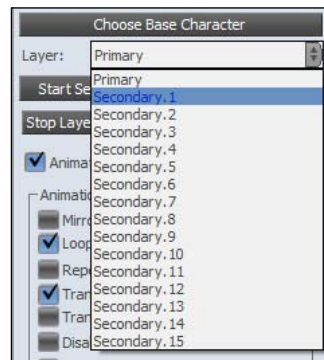
1. To play an animation for a currently loaded model, simply browse to the animation name and click it and the character will immediately start the animation.
2. For this example, start the animation `_Relaxed_Run` under the `_Relaxed` folder.



3. You will notice the **LMG** icon located beside the animation name. These will be discussed later. It is, however, important to know that LMGs are sets of animations which are procedurally blended between.
4. Ensure that Animation Driven Motion is set to `true`.
5. Adjust the first slider in the blend space control to `-1`.



6. You will notice that the character slows down and blends naturally between a fast and slow run. The other sliders control the rotation and the incline.
7. Reset all the sliders to 0.
8. Now, we will play an animation in a different layer on the same character. This can be useful for upper body-only animations, which we will create in a recipe.
9. While the character is playing the `_Relaxed_Run` LMG animation, change the **Layer** property to **Secondary.1**.



10. Next, select the animation under the throw folder `throw_leftarm_UB_01`.
11. You will see that the character begins to play both animations. The second animation is being played on the second layer in a layered fashion.



Layer zero is the primary-layer. Usually it contains the base full body animation. All animation in layer zero must be set to full body animations.

How it works...

The animation system in the CryENGINE is a combination of a variety of existing skeleton animation technologies, including playback and blending of animation data, as well as IK-based pose-modifications. Procedural algorithms such as CCD-IK, analytic IK, example-based IK, or physical simulations are sometimes used to augment pre-authored animations.

- ▶ The Character Editor allows character, artists, and animators to see all of these systems interacting with their content and thus, they are able to modify the content quickly and preview it to ensure its quality.
- ▶ We explored the layer system which allows us to apply an animation to only a few selected bones, rather than to the whole skeleton.
- ▶ The Layer System can support up to 16 virtual layers. Layered Animations are all applied at the same time onto a character.
- ▶ This system was designed mainly for mechanical objects (such as weapons, vehicles, and so on). However, in some cases it can be used on humans as well. Human animations in a game often consist of a base animation (standing, walking, and running) and a number of modified derivatives (looking around, aiming with a gun, and reloading a gun).
- ▶ Combining animations is equally useful for animating characters that have a large variety of equipment or weapons, while reusing basic cycles such as walks and runs.

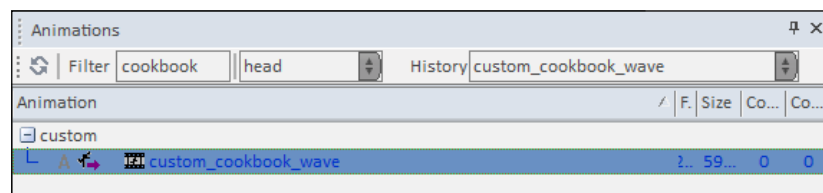
There's more...

You will find it useful to search and filter animations as well as know the different types of animation assets. You can find out more about these topics in the following sections.

Animation driven motion

This flag is set per character instance. Set this flag which will then extract the real movement (translation and rotation per frame) of the character from the locomotion animation file and pass this information to the game-code to move the character in the world. This feature requires specifically authored motion-assets with a locomotion locator. A locomotionlocator describes the logical movement of the motionclip in the simplest way.

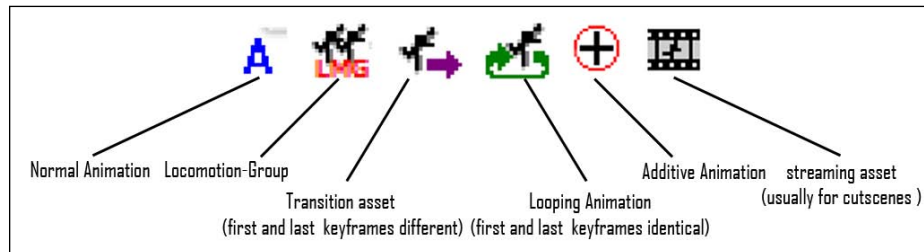
Searching and filtering animations



Using the filter in the animation list is a good technique to quickly find particular animations. There is also a history window, which you can use to quickly move between recently played animations.

Types of animation assets

Animation assets will be tagged with a certain icon depending on the type of content.



See also

- ▶ It is important that you have animations to work with. Refer to the *Creating animation for your character* recipe mentioned earlier in this chapter
- ▶ To create upper body animations refer to the next recipe in this chapter

Creating upper body only animations

The easiest way to create assets for partial-body animation is to remove the animation channels for certain bones in the DCC App. That means that you have to create a set of special animations that influence just a limited set of bones and not the entire skeleton.

Getting ready

To complete this recipe, you must finish the *Creating animation for your character* recipe mentioned in the earlier sections of this chapter.

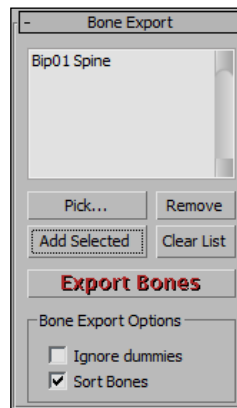
How to do it...

Let's create an upper body only animation:

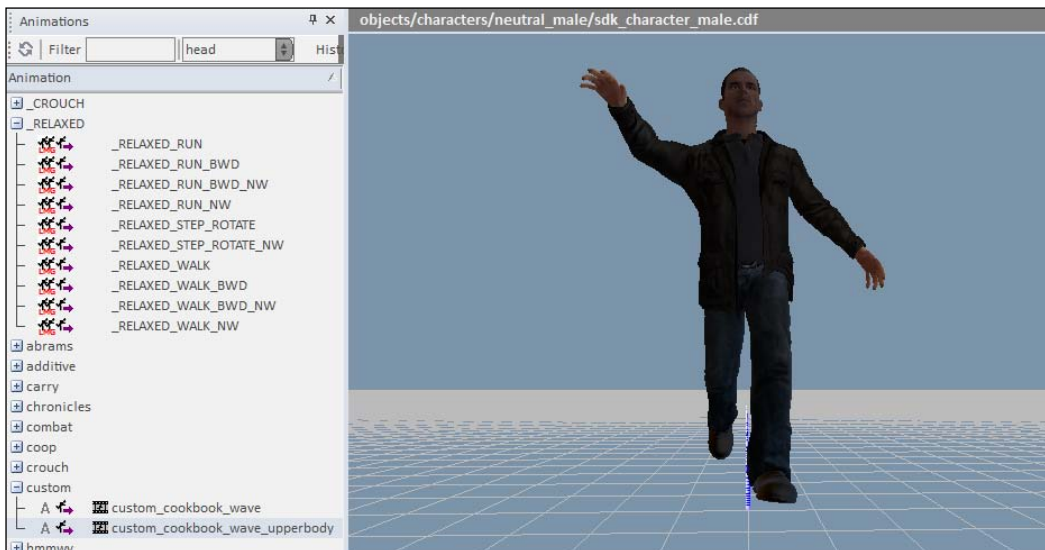
- ▶ Open up the animation `cookbook_wave` animation.
- ▶ When we last exported this animation, we exported it as a full body animation. Exporting an upper body animation is much the same as exporting a full body animation. You need to select a root of a hierarchy in the skeleton and add that to the Export Bones list manually.

- ▶ Select the **Bip01 Spine** bone and then click **Clear List** in the **Bone Export** options and then **Add Selected**. This will add the bone Bip01 Spine to the node list.

You are now ready to export the animation from the Bip01 Spine bone and all of its children in the hierarchy.



1. Click the export bones button and save the animation as `cookbook_wave_upperbody.caf`.
2. Start the animation `_Relaxed_Run` under the `_Relaxed` folder in the character editor.
3. Change the layer to the secondary layer and play the newly exported upper body animation on the second layer.



4. This will begin to play the animation using the tracks that were exported from the spine down through its hierarchy.

How it works...

You should now see the combined movement. Notice the legs moving in the same way as the full body animation, with the upper body completely overwritten by the second animation.

There's more...

You may also want to know how additive animations work; these are similar to upper body animations and can be used to play facial animation.

Additive animations

Additive animations are relative animations, which mean they don't overwrite the animation on the original bones, but instead add motion to them.

This means that an additive animation will preserve the underlying animation and style, which is great for adding poses and animations.

This can reduce the overall asset count greatly and add a lot of variation to the animations.

Possible uses of additives could be breathing, looking around, flinching, posture change, and so on.

Using additives

An additive can be started like a regular animation, as the engine will automatically recognize it as an additive animation after it has been processed by the resource compiler.

See also

- ▶ To learn how to create locomotion animations such as the one used in the previous recipe, refer to the *Creating locomotion animations* recipe in this chapter
- ▶ You will likely need to know how to create an animation for your character and the principles behind it for this recipe. Please refer to the *Creating animation for your character* recipe mentioned earlier in this chapter

Creating locomotion animations

Locomotion animations are absolutely essential for AI and player actions and animations.

Authoring these assets can be easy if a few principles are understood regarding a very important rig element called the `Locator_Locomotion`.

The `locomotion locator` or `Locator_Locomotion` as the node is called, is used in the engine to describe the logical movement and orientation of the animation. This node needs to be added to motions which translate in a non uniform way, such as a start or stop transition, which have peaks and troughs in acceleration.

Getting ready

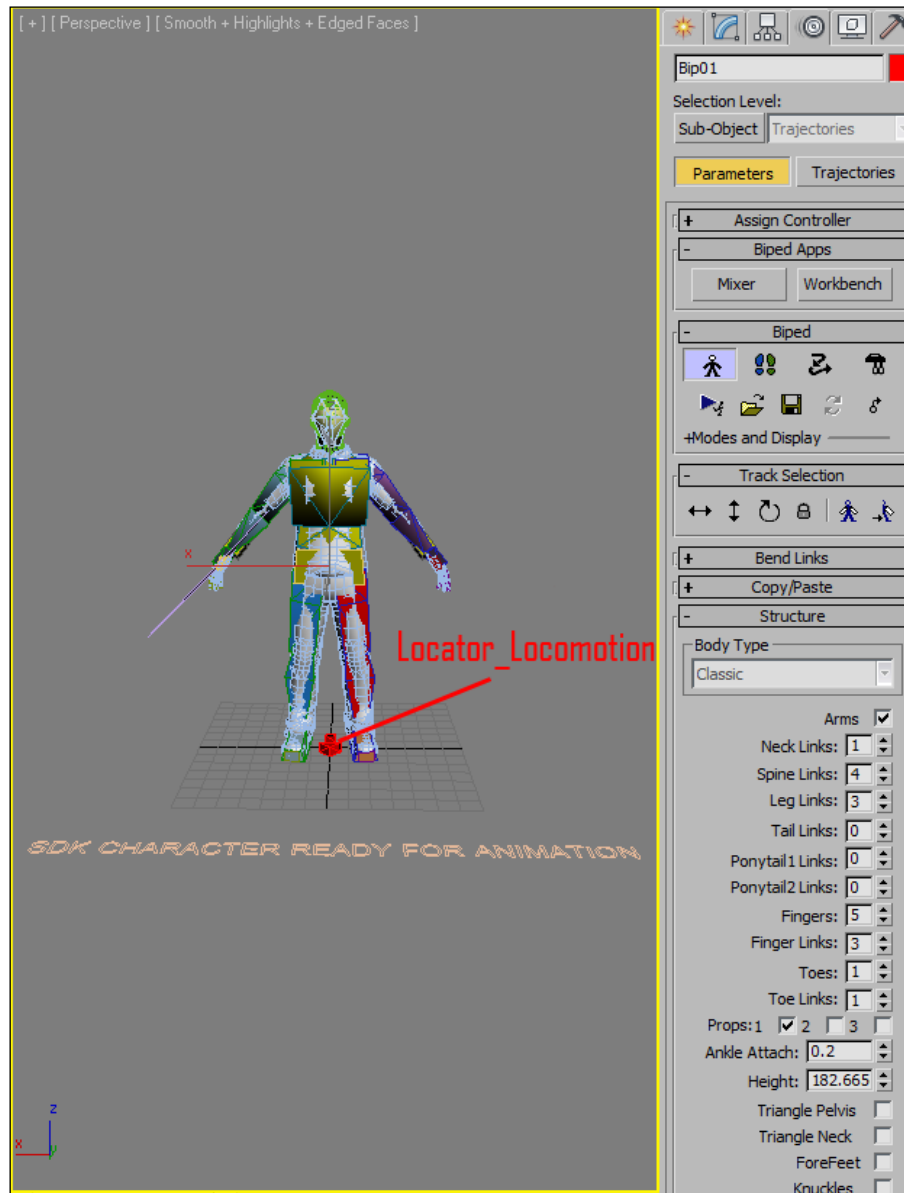
- ▶ The skeleton must contain the bone `Locator_Locomotion` if it is going to be used in a LMG group later
- ▶ Open the file `SDK_character_male_foranimation.max`

How to do it...

Let's learn how to create the structure needed to support locomotion animations:

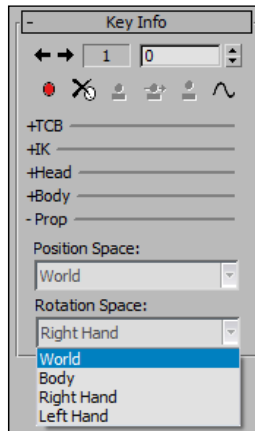
1. When using a **biped**, the easiest way to add the ability to use locomotion is to turn on figure mode and create a prop named `Locator_Locomotion`.
2. In this way, the animation for the locator can be saved inside the biped file, making it easier for exporting.

3. Simply access figure mode, go to the **structure** menu of **biped**, and check the box **Props:1**:



4. Make sure the bone faces the positive Y axis in its local coordinate system. We will now begin animating the locator.

5. Start by setting a key for the prop, then navigate to the prop menu under the **Key Info** group of the biped motion menu and set both **Position Space** and **Rotation Space** to world. This will allow you to animate the node independent of the root of the character.



6. When creating an animation in which the character has to turn quickly as in an idle to run transition, the locator needs to be a projection of the **Bip01** on the ground, yet it must travel in a straight line on one axis. Its height must be set to 0.
7. Keep in mind that the locator should be moving in a straight line in the direction with no deviation to either side. However, the actual translation of the locator in its forward moving direction should follow the **Bip01** as closely as possible.

How it works...

Properly setting up locomotion animations allows the use of the procedural blend space system within CryENGINE. This system uses LMGs or locomotion groups.

An LMG is an XML file containing a blend-code and a list of animation names in a fixed order:

```
<LocomotionGroup>
  <BLENDTYPE type="STF2" />
  <CAPS code="RUN" />

  <ExampleList>
    <Example Position=" 0, 1, 0" AName="combat_run_rifle_forward_fast"
  />
    <Example Position="-1, 0, 0" AName="combat_run_rifle_left_fast" />
    <Example Position=" 0,-1, 0" AName="combat_run_rifle_back_fast" />
    <Example Position=" 1, 0, 0" AName="combat_run_rifle_right_fast"
  />
```



```
<Example Position=" 0, 1, 0" AName="combat_run_rifle_forward_slow"
/>
<Example Position="-1, 0, 0" AName="combat_run_rifle_left_slow" />
<Example Position=" 0,-1, 0" AName="combat_run_rifle_back_slow" />
<Example Position=" 1, 0, 0" AName="combat_run_rifle_right_slow"
/>
</ExampleList>

</LocomotionGroup>
```

The blend-code is used at loading time to identify the type of LMG, to extract the features out of the root-motion, to verify the assets, and to compute the blend-values at runtime. The game simply specifies a parameter and the animation system generates the desired motion. The motion features are extracted directly out of the clips.

The idea behind locomotion groups is to collect multiple variations of a motion-type and arrange them in a spatial data-structure, which in CryENGINE is called **blendspace**. At runtime, it then applies interpolation and extrapolation techniques to create an infinite number of motions between and around these example motions.

Animating human figures can be very complex. However, it is possible to describe many actions with a relatively small number of motion-parameters. For example, a locomotion cycle might be characterized by the speed, body-orientation, travel-direction, turn-radius, and slope-angle. Such kinematic and physical properties can be combined with abstract properties like mood or style.

While a programmer is not required to use existing LMG blend-codes, programmers are needed for their creation. Fortunately, LMG blend-codes already exist for most locomotion purposes.

There's more...

Having created some basic locomotion, you may want to know some specifics when creating swimming animations or locomotion loops.

Swimming and vehicle transitions

For swimming transitions or vehicle transitions the locator can be a straight blend between the ground position of 0, 0, z and end at the **Bip01** location and forward-looking direction of the character.

Locomotion loops

For loops, only setting keys for the start and end of the animation is necessary, if you wish to add a locator to them. They are technically not needed, but for batch processing it keeps the database clean.

Idle to move and 180 degree rotational assets

The orientation of the locator in an idle mode to move transition should remain looking forward until frame 10.

When the orientation changes (left, right, left reverse, or right reverse) this should occur in the following 6 frames, so the new orientation is complete at frame 16.

When changing the orientation 180 degrees for reverse transitions, make sure you rotate the locator 0.1 degrees back to its original orientation to avoid flipping of the character.

See also

- ▶ You must know how to animate a character to be able to create locomotion animations. Refer to the *Animating your character in CryENGINE* recipe mentioned in the earlier sections of this chapter
- ▶ If you haven't yet created a character then refer to the *Creating a skinned mesh* recipe mentioned in the earlier sections of this chapter

Animating rigid body geometry data

In the CryENGINE, not all animation needs to be driven by bones. The engine supports animating hard body geometry, which can be far more optimal for certain assets and in certain situations. You might want to animate complex scenes such as the destruction of an object in 3ds Max and then export that to the engine, or you might simply want to open the door on a vehicle—all this can be done using rigid body geometry animation.

Getting ready

Animated hard body geometry data only supports directly linked objects and does not support skeletal-based animation. It is composed of two file types:

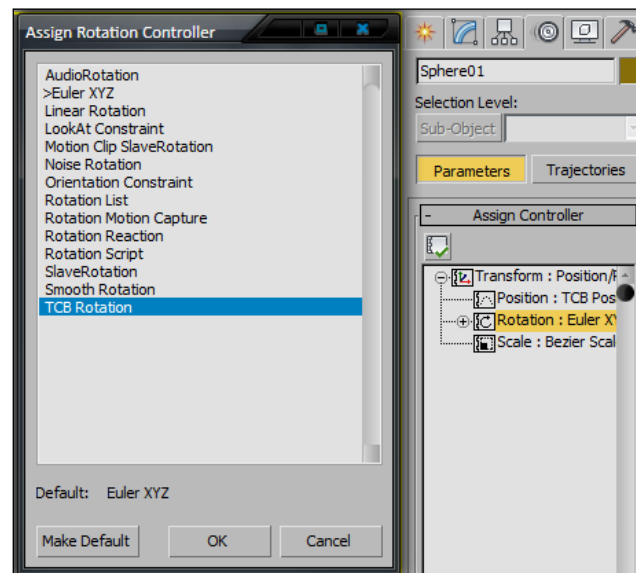
- ▶ `.cga` (Crytek Geometry Animation)
- ▶ `.anm` (Additional Animation)

How to do it...

Let's create some rigid body animation:

1. In 3ds, create two simple spheres primitives in the scene.
2. Convert the primitives to edit polys.
3. As we will be exporting a `.anm` file for a `.cga` geometry, we must convert all the objects that will be animated to the Tension Continuity and Bias controller or TCB.

4. Change the controller types for position and rotation to TCB under the **3ds Motion** tab.



5. Now, create a very simple animation for the objects in the scene. Bouncing the spheres is a simple animation to make.
6. Next, we must export the geometry before exporting the animation for it.
7. Select all the objects and add them to the **Object Export** list by clicking **add selected**.
8. For this example, make sure export file per node is set to `false` as we would like to have both the geometric objects in a single `.cga` file.
9. Next, make sure that the `cga` file type is selected in the **Export To** drop-down box.
10. Save the max file as `cookbook_bouncing_balls.max` under the `game/objects/bouncingballs` folder.
11. Press the **Export Nodes** button. This will export the objects to the same folder that the source file is located.
12. The file will assume the source file's name unless you check **Custom Filename**. This will allow you to type in a unique name.
13. Next, for good workflow, we should examine our newly created `.cga` in the character editor.
14. Open the CryENGINE Sandbox and the character editor.
15. Click **File | Open** and navigate to the folder you exported your `.cga` file to.
16. Each CGA file contains a default animation, and it is the length of the timeline that was active when the `.cga` file is exported.

17. We now have an existing CGA file that we want to add with a selectable animation to create a second animation for the .cga.
18. When the animation is complete, navigate to the **Object Export** section of the exporter and set **Custom Filename** to **true** and set a filename that uses the name of the .cga object you are adding it to, as a prefix in the name.
19. As an example, the test .cga file contains the object you are animating, in addition to the default animation. In this case, you need to name your .anm file test_[n] .anm, where n questions the name of the additional animation.
20. Press the **Export Nodes** button.
21. The .anm file must be saved to the same folder as the .cga; otherwise, you will not be able to preview it in the character editor.

How it works...

.anm files are the simplest animation that you can export to the CryENGINE. They do not require as much processing power as a skinned mesh and can be animated fairly easily. These types of animations can be used on a variety of environmental objects that require some moving parts. The .anm file format is also the principal animation format for any vehicle animation.

There's more...

You may want to know how to bake physically simulated animations into .cga animations and how to use an animobject entity.

Pre-baked physics with .CGA objects

- ▶ To pre-bake physical destruction, you can use .cga objects with baked .anm animations.
- ▶ This can be very useful for complex destruction of large objects.
- ▶ The same rules apply when creating pre-baked physics for .cga objects, as all the position and rotation controllers of the objects in the CGA need to be set to TCB.
- ▶ When exporting pre-baked physics animations select **merge all nodes** in the **export** options.
- ▶ Turn off all **Bone Export** options, but tell the exporter to export every 1 frames.
- ▶ Select one object that is not moving and parent all the other pieces to it. This object can be one you set to **unyielding**. In buildings, use the foundation.
- ▶ The current animation will show up as **Default** in the CGA you export. No CAL file is needed to play it.
- ▶ Open your CGA in the character editor and play the animation labeled **Default**. You should see your animation play.

Anim object and pre-baked .CGA

When using the pre-baked .cga in the game environment, it must be placed as an animobject.

There are two important properties available when using pre-baked .cga:

- ▶ The first is that ActivatePhysicsThreshold is a fraction of levels gravity; thus, a heavier piece will be harder to activate since the gravitational force that acts on it is stronger.
- ▶ The second is the mass. Mass is set as the overall value for the entire CGA; for instance, a Mass of 100 on a CGA with 100 pieces would yield 1kg per piece.

See also

- ▶ You may want to create a vehicle to animate. Refer to the *Creating a new car mesh* recipe in *Chapter 8, Creating Vehicles*

8

Creating Vehicles

In this chapter, we will cover:

- ▶ Creating a new car mesh (CGA)
- ▶ Creating a new car XML
- ▶ Giving more speed to the car
- ▶ Increasing the mass to push objects with the car
- ▶ Defining a sitting location
- ▶ Setting up multiple cameras for the car
- ▶ Need for a **machine gun**
- ▶ Giving the car a weak spot

Introduction

By this point, we have covered a vast majority of the basic principles behind a lot of the main components of **CryENGINE 3**. For this chapter, we will explore some more advanced components by creating a new vehicle for your player to drive around in your level.

Creating a new car mesh (CGA)

In this recipe, we will show you how to build the basic mesh structure for your car to be used in the next recipe. This recipe is not to be viewed as a guide on how to model your own mesh, but rather as a template for how the mesh needs to be structured to work with the XML script of the vehicle. For this recipe, you will be using 3DSMax to create and export your .CGA.



.CGA (Crytek Geometry Animation): The .cga file is created in the 3D application and contains animated hard body geometry data. It only supports directly-linked objects and does not support skeleton-based animation (bone animation) with weighted vertices. It works together with .anim files.

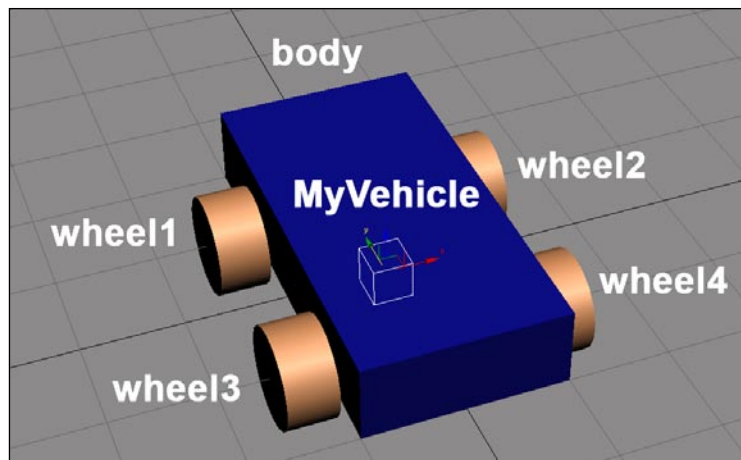
Getting ready

Review the *Creating and exporting the static objects* recipe of *Chapter 6, Asset Creation*. Create a box primitive and four cylinders within Max and then create a new dummy helper.

How to do it...

After creating the basic primitives within Max, we need to rename these objects. Rename the primitives to match the following naming convention:

- ▶ Helper = **MyVehicle**
- ▶ Box = **body**
- ▶ Front Left Wheel = **wheel1**
- ▶ Front Right Wheel = **wheel2**
- ▶ Rear Left Wheel = **wheel3**
- ▶ Rear Right Wheel = **wheel4**



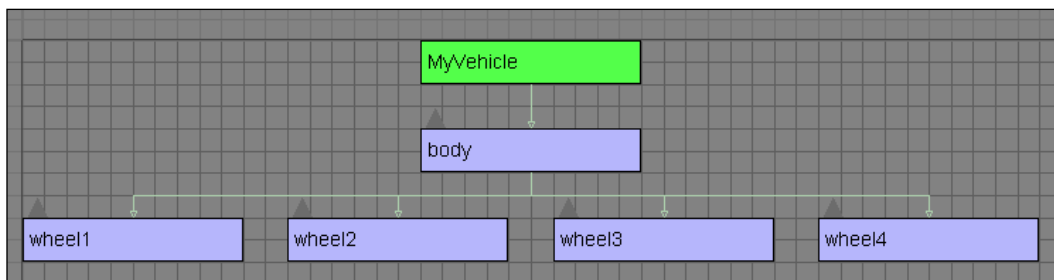


Remember that CryENGINE 3 assumes that y is forward. Rotate and reset any x-forms if necessary.

From here you can now set up the hierarchy to match what we will build into the script:

1. In Max, link all the **wheels** to the **body** mesh.
2. Link the **body** mesh to the **MyVehicle** dummy helper.

Your hierarchy should look like the following screenshot in the Max schematic view:

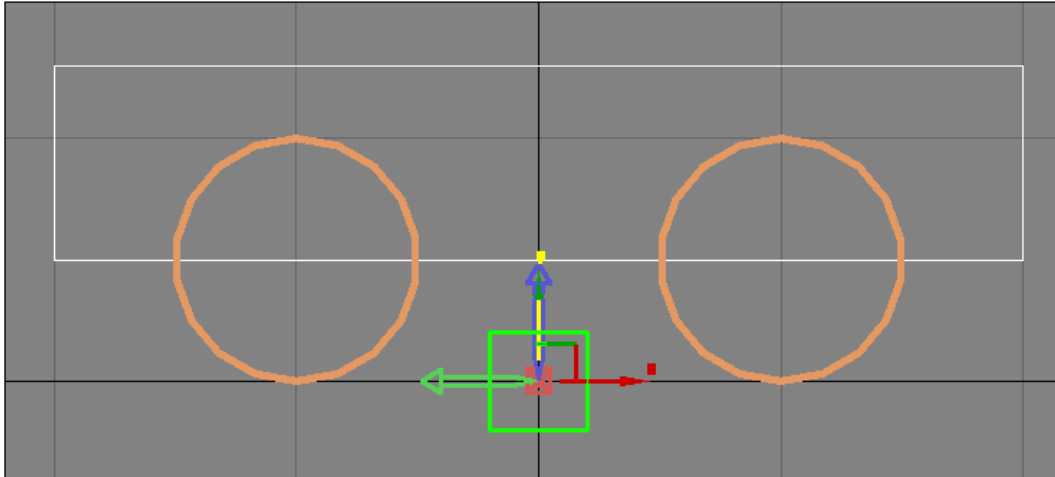


Next, you will want to create a *proxy mesh* for each *wheel* and the *body*. Be sure to attach these proxies to each mesh. Proxy meshes can be a direct duplication of the simple primitive geometry we have created.

Before we export this mesh, make one final adjustment to the positioning of the vehicle:

1. Move the body and the wheels up on the **Z** axis to align the bottom surface of the wheels to be flushed with **0** on the **Z**.
2. Without moving the body or the wheels, be sure that the **MyVehicle** helper is positioned at **0,0,0** (this is the origin of the vehicle).
3. Also, re-align the pivot of the **body** to **0,0,0**.

Once complete, your left viewport should look something like the following screenshot (if you have your **body** still selected):

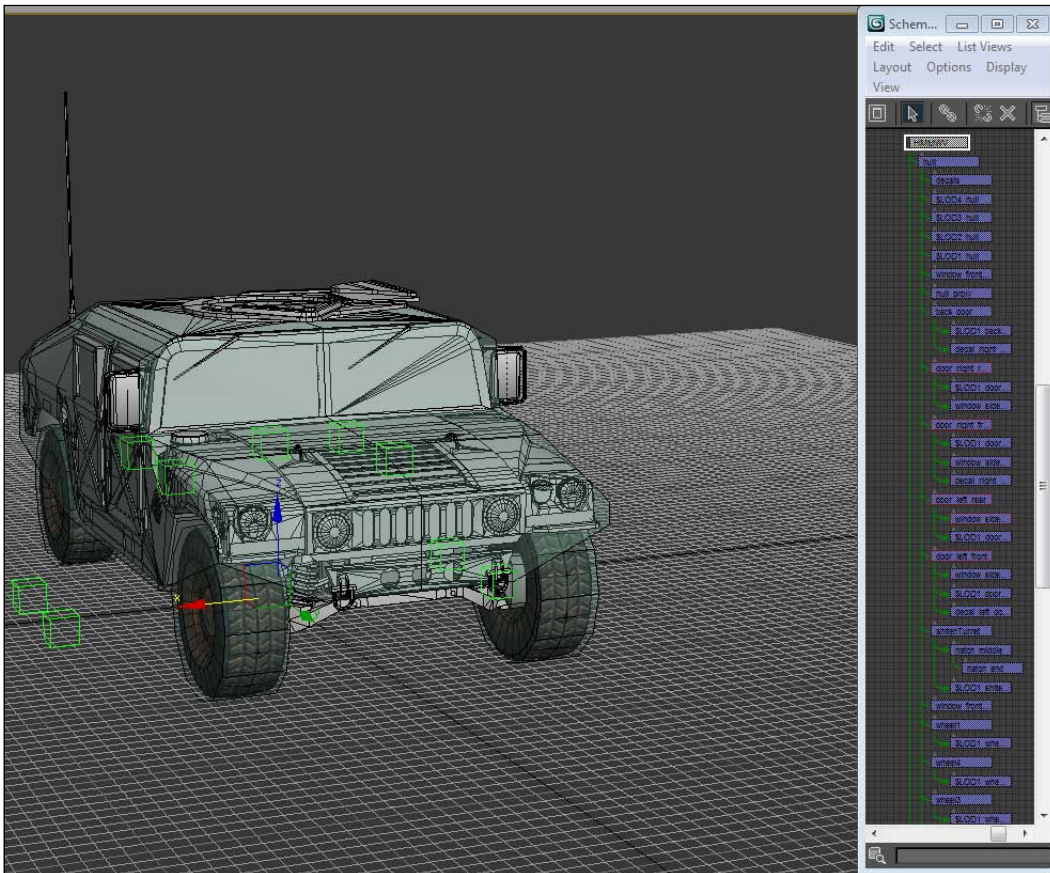


After setting up the materials that you learned from *Chapter 6, Asset Creation*, you are now ready to export the CGA:

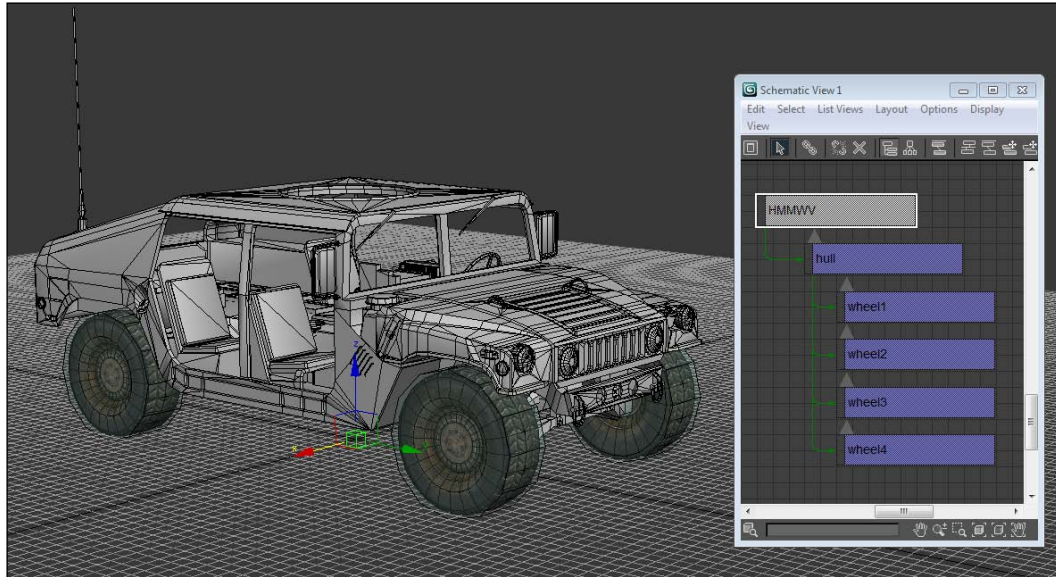
1. Open the **CryENGINE Exporter** from the **Utilities** tab.
2. Select the **MyVehicle** dummy helper and click the **Add Selected** button.
3. Change the export to: **Animated Geometry (*.cga)**.
4. Set **Export File per Node** to **True**.
5. Set **Merge All Nodes** to **False**.
6. Save this Max scene in the **following directory**: `MyGameFolder\Objects\vehicles\MyVehicle\`.
7. Now, click on **Export Nodes** to export the CGA.

How it works...

This setup of the CGA is a basic setup of the majority of the four wheeled vehicles used for CryENGINE 3. This same basic setup can also be seen in the HMMWV provided in the included assets with the SDK package of CryENGINE 3.



Even though the complete HMMWV may seem to be a very complicated mesh used as a vehicle, it can also be broken down into the same basic structure as the vehicle we just created.



The main reason for the separation of the parts on the vehicles is because each part performs its own function. Since the physics of the vehicle code drives the vehicle forward in the engine, it actually controls each wheel independently, so it can animate them based on what they can do at that moment. This means that you have the potential for a four wheel drive on **all CryENGINE 3 vehicles, all animating at different speeds based on the friction that they grip.**

Since all of the wheels are parented to the body (or hull) mesh, this means that they drive their parent (the body of the vehicle) but the body also handles where the wheels need to be offset from in order to stay aligned when **driving. The body itself acts as the base mesh for all other extras put onto the vehicle.** Everything else from *Turrets* to *Doors* to *Glass Windows* branch out from the body.

The dummy helper is only the parent for the body mesh due to the fact that it is easier to export multiple LODs for that vehicle (for example, HMMWV, HMMWV_LOD1, HMMWV_LOD2, and so on). In the XML, this dummy **helper is ignored in the hierarchy and the body** is treated as the parent node.

There's more...

Here are some of the more advanced techniques used.

Dummy helpers for modification of the parts

A more advanced trick is the use of dummy helpers set inside the hierarchy to be used in later reference through the vehicle's mod system. How this works is that if you had a vehicle such as the basic car shown previously, but you wanted to add on an additional mesh just to have a modified type of this same car (something like adding a spoiler to the back), then you can create a dummy helper and align it to the pivot of the object, so it will line up to the body of the mesh when added through the script later on.

This same method was used in *Crysis 2* with the Taxi signs on the top of the Taxi cars. The Taxi itself was the same model used as the basic civilian car, but had an additional dummy helper where the sign needed to be placed. This allowed for a clever way to save on memory when rendering multiple vehicle props within a single area but making each car look different.

Parts for vehicles and their limitless possibilities

Adding the basic body and four wheels to make a basic car model is only the beginning. There are limitless possibilities to what you can make as far as the parts on a vehicle are concerned. Anything from a classic gunner turret seen on the HMMWV or even tank turrets, all the way to arms for an articulated Battlemech as seen in the *Crysis 2 Total Conversion mod—MechWarrior: Living Legends*. Along with the modifications system, you have the capabilities to add on a great deal of extra parts to be detached and exploded off through the damage scripts later on. The possibilities are limitless.

See also

- ▶ *Creating a new car XML*
- ▶ The *Creating and exporting the static objects* recipe in *Chapter 6, Asset Creation*

Creating a new car XML

In this recipe, we will show you how to build a new script for **CryENGINE 3** to recognize your car model as a vehicle entity. For this recipe, you must have **some basic knowledge** in XML formatting.

Getting ready

Open `DefaultVehicle.xml` in the XML editor of your choice (Notepad, Notepad++, UltraEdit, and so on). This XML will be used as the basic template to construct our new vehicle XML.



`DefaultVehicle.xml` is found at the following location: `MyGameFolder\Scripts\Entities\Vehicles\Implementations\Xml`.

Open the `MyVehicle.max` scene made from the previous recipe to use as a reference for the parts section within this recipe.

How to do it...

Basic Properties:

1. First, we will need to rename the **filename** to what the vehicle's name would be.
2. Delete **filename = Objects/Default.cgf**.
3. Rename **name = DefaultVehicle** to **name = MyVehicle**.
4. Add **actionMap = landvehicle** to the end of the cell.
5. Save the file as `MyVehicle.XML`.
6. Your first line should now look like the following:



Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

```
<Vehicle name="MyVehicle" actionMap="landvehicle">
```

7. Now we need to add some physics simulation to the vehicle otherwise there might be some strange reactions with the vehicle. Insert the following after the third line (after the Buoyancy cell):

```
<Simulation maxTimeStep="0.02" minEnergy="0.002"
maxLoggedCollisions="2"/>
```

Damages and Components:

For now, we will skip the Damages and Components cells as we will address them in a different recipe.

Parts:

1. To associate the parts made in the Max file, the hierarchy of the geometry in 3DSMax needs to be the very same as is referenced in the XML. To do this, we will first clear out the `class = Static` cell and replace it with the following:

```
<Part name="body" class="Animated" mass="100" component="Hull">
  <Parts>
  </Parts>
  <Animated filename="objects/vehicles/MyVehicle/MyVehicle.cga"
filenameDestroyed="objects/vehicles/HMMWV/HMMWV_damaged.cga"/>
</Part>
```

2. Now, within the `<Parts>` tag that is underneath the body, we will put in the wheels as the children:

```
<Parts>
  <Part name="wheel1" class="SubPartWheel" component="wheel_1"
mass="80">
    <SubPartWheel axle="0" density="0" damping="-0.7" driving="1"
lenMax="0.4" maxFriction="1" minFriction="1" slipFrictionMod="0.3"
stiffness="0" suspLength="0.25" rimRadius="0.3"
torqueScale="1.1"/>
  </Part>
</Parts>
```

3. Remaining within the `<Parts>` tag, add in wheels 2-4 using the same values as previously listed. The only difference is you must change the `axle` property of wheels 3 and 4 to the value of 1 (vehicle physics has an easier time calculating what the wheels need to if only two wheels are associated with a single axle).
4. The last part that needs to be added in is the **Massbox**. This part isn't actually a mesh that was made in 3DSMax, but a generated bounding box, generated by code with the mass and size defined here in the XML. Write the following code snippet after the `<body>` tag:

```
<Part name="massBox" class="MassBox" mass="1500" position="0,0,1."
disablePhysics="0" disableCollision="0" isHidden="0">
  <MassBox size="1.25,2,1" drivingOffset="-0.7"/>
</Part>
```

5. If scripted correctly, your script should look similar to the following for all of the parts on your vehicle:

```
<Parts>
  <Part name="body" class="Animated" mass="100" component="Hull">
    <Parts>
      <Part name="wheel1" class="SubPartWheel" component="wheel_1"
mass="80">
        <SubPartWheel axle="0" density="0" damping="-0.7"
```

```

driving="1" lenMax="0.4" maxFriction="1" minFriction="1"
slipFrictionMod="0.3" stiffness="0" suspLength="0.25"
rimRadius="0.3" torqueScale="1.1"/>
</Part>
<Part name="wheel2" class="SubPartWheel" component="wheel_2"
mass="80">
  <SubPartWheel axle="0" density="0" damping="-0.7"
driving="1" lenMax="0.4" maxFriction="1" minFriction="1"
slipFrictionMod="0.3" stiffness="0" suspLength="0.25"
rimRadius="0.3" torqueScale="1.1"/>
</Part>
<Part name="wheel3" class="SubPartWheel" component="wheel_3"
mass="80">
  <SubPartWheel axle="1" density="0" damping="-0.7"
driving="1" lenMax="0.4" maxFriction="1" minFriction="1"
slipFrictionMod="0.3" stiffness="0" suspLength="0.25"
rimRadius="0.3" torqueScale="1.1"/>
</Part>
<Part name="wheel4" class="SubPartWheel" component="wheel_4"
mass="80">
  <SubPartWheel axle="1" density="0" damping="-0.7"
driving="1" lenMax="0.4" maxFriction="1" minFriction="1"
slipFrictionMod="0.3" stiffness="0" suspLength="0.25"
rimRadius="0.3" torqueScale="1.1"/>
</Part>
</Parts>
<Animated filename="objects/vehicles/MyVehicle/MyVehicle.cga"
filenameDestroyed="objects/vehicles/HMMWV/HMMWV_damaged.cga"/>
</Part>
<Part name="massBox" class="MassBox" mass="1500"
position="0,0,1." disablePhysics="0" disableCollision="0"
isHidden="0">
  <MassBox size="1.25,2,1" drivingOffset="-0.7"/>
</Part>
</Parts>

```

Movement Parameters:

Finally, you will need to implement the `MovementParams` needed, so that the XML can access a particular movement behavior from the code that will propel your vehicle. To get started right away, we have provided an example of the `ArcadeWheeled` parameters, which we can copy over to `MyVehicle`:

```

<MovementParams>
  <ArcadeWheeled>
    <Steering steerSpeed="45" steerSpeedMin="80" steerSpeedScale="1"
steerSpeedScaleMin="1" kvSteerMax="26" v0SteerMax="40"

```

```

steerRelaxation="130" vMaxSteerMax="12"/>
  <Handling>
    <RPM rpmRelaxSpeed="2" rpmInterpSpeed="4"
rpmGearShiftSpeed="2"/>
    <Power acceleration="8" deceleration="0.1" topSpeed="32"
reverseSpeed="5" pedalLimitMax="0.30000001"/>
    <WheelSpin grip1="5.75" grip2="6" gripRecoverSpeed="2"
accelMultiplier1="1.2" accelMultiplier2="0.5"/>
    <HandBrake deceleration="15" decelerationPowerLock="1"
lockBack="1" lockFront="0" frontFrictionScale="1.1"
backFrictionScale="0.1" angCorrectionScale="5" latCorrectionScale="1"
isBreakingOnIdle="1"/>
    <SpeedReduction reductionAmount="0" reductionRate="0.1"/>
    <Friction back="10" front="6" offset="-0.2"/>
    <Correction lateralSpring="2" angSpring="10"/>
    <Compression frictionBoost="0" frictionBoostHandBrake="4"/>
  </Handling>
  <WheeledLegacy damping="0.11" engineIdleRPM="500"
engineMaxRPM="5000" engineMinRPM="100" stabilizer="0.5"
maxTimeStep="0.02" minEnergy="0.012" suspDampingMin="0"
suspDampingMax="0" suspDampingMaxSpeed="3"/>
  <AirDamp dampAngle="0.001,0.001,0.001" dampAngVel="0.001,1,0"/>
  <Eject maxTippingAngle="110" timer="0.3" />
  <SoundParams engineSoundPosition="engineSmokeOut"
runSoundDelay="0" roadBumpMinSusp="10" roadBumpMinSpeed="6"
roadBumpIntensity="0.3" maxSlipSpeed="11"/>
</ArcadeWheeled>
</MovementParams>

```

After saving your XML, open the Sandbox Editor and place down from the **Entities** types: Vehicles\MyVehicle. You should now be able to enter this vehicle (get close to it and press the *F* key) and drive around (*W* = accelerate, *S* = brake/reverse, *A* = turn left, *D* = turn right)!



How it works...

The parts defined here in the XML are usually an exact match to the Max scene that the vehicle is created in. As long as the naming of the parts and the name of the subobjects within Max are the same, the vehicle structure should work.

The parts in the XML can themselves be broken down into their own properties:

- ▶ Name: **The name of the part.**
- ▶ Class: **The classification of the part.**
- ▶ Base (obsolete)
- ▶ Static: Static vehicle (should not be used).
- ▶ Animated: The main part for an active rigid body of a vehicle.
- ▶ AnimatedJoint: Used for any other part that's used as a child of the animated part.
- ▶ EntityAttachment (obsolete)
- ▶ Light: Light parts for headlights, rear light, and so on.
- ▶ SubPart (obsolete)
- ▶ SubPartWheel: Wheels.
- ▶ Tread: Used with tanks.
- ▶ MassBox: Driving Massbox of the vehicle.
- ▶ Mass: **Mass of the part (usually used when the part is detached)**
- ▶ Component: **Which component this part is linked to. If the component uses useBoundsFromParts="1", then this part will also be included in the total bounding box size.**
- ▶ Filename: **If a dummy helper is created in Max to be used as a part, then an external mesh can be referenced and used as this part.**
- ▶ DisablePhysics: **Prevents the part from being physicalized as rigid.**
- ▶ DisableCollision: **Disables all collision. It is an useful example for mass blocks.**
- ▶ isHidden: **Hides the part from rendering.**

There's more...

The `def_vehicle.xml` file found in `MyGameFolder\Scripts\Entities\Vehicles`, holds all the property's definitions that can be utilized in the XML of the vehicles. After following the recipes found in this chapter, you may want to review `def_vehicle.xml` for further more advanced properties that you can add to your vehicles.

See also

- ▶ *Giving more speed to the car*
- ▶ *Increasing the mass to push objects with the car*
- ▶ *Giving the car a weak spot*
- ▶ *Defining a sitting location*

Giving more speed to the car

Now that we have created a basic template for a car within the game, we can now start manipulating more of the fun properties that the XML holds. Let us start out by giving this car a bit more speed.

Getting ready

Complete the *Creating a new car XML* recipe. Then **open** `MyVehicle.xml` in Notepad or an equivalent editor.

How to do it...

Under the `<ArcadeWheeled>` cell, you will find the `<Power>` tag; within the `<Power>` tag you will find a property called `topSpeed=32`. It is a simple matter of increasing this value to increase the car's overall top speed.

How it works...

The *Arcade Wheeled* movement property is a new movement behavior **since CryENGINE 3**. Dealing away with the confusing gear ratios, new backend code has been written **to create** the *Arcade Wheeled* for the purpose of being able to tweak the major values on wheeled vehicles much easier.

There's more...

The following are the common properties of an *Arcade Wheeled*:

- ▶ **Acceleration:** How fast the vehicle will speed up
- ▶ **Deceleration:** How fast the vehicle will slow down
- ▶ **TopSpeed:** Top speed that the vehicle can achieve
- ▶ **ReverseSpeed:** Reversing speed
- ▶ **Handbrake-deceleration:** Deceleration when hand brake is applied

See also

- ▶ *Creating a new car XML*
- ▶ *Increasing the mass to push objects with the car*
- ▶ *Giving the car a weak spot*

Increasing the mass to push objects with the car

In this recipe, we explore some of the possibilities of manipulating the Massbox that we created when first creating the car's XML. With increasing the Massbox of the car, we will see how the car is able to push lighter objects out of the way.

Getting ready

Complete the *Creating a new car XML* recipe. Then **open** `MyVehicle.xml` in Notepad or an equivalent editor.

How to do it...

As simple as the previous recipe was, this one will be just as easy to make a massive change to your vehicle.

Begin by finding the car's part named `massBox`. To increase the mass of the MassBox, simply increase the `mass = 1500` to a higher value such as 3500.



The mass in CryENGINE 3 is measured in kilograms.

How it works...

A simple method to see the results of how easy it is for the car to move objects out of its way is to place down a basic entity with a mass of 1000 and then drive into it with the previous MassBox properties. Then repeat the process again with the 3500 MassBox value.



After changing any XML values on the car, you can reload the XML on the car by right-clicking on the vehicle and selecting **Reload Scripts**.

The MassBox is a special part of the vehicle that handles all of the physical interactions of the vehicle as a whole. Along with Arcade Wheeled as the movement behavior, the Massbox is the main point of reference for the car and all its parts for mass physics calculations (as long as all those parts remain on the vehicle).



Be aware that increasing the mass of the MassBox will also change the driving characteristics of the car.

There's more...

It is possible to define the mass of other parts besides the Massbox. However, with Arcade Wheeled movement parameters, this property change will do virtually nothing, but with other movement behaviors it may change the driving mechanics such as the *Wheeled Legacy* (which should not be used as that movement behavior is no longer supported).

Still, there is a use of this property and that is if ever we want to create a part to be detached from the vehicle, this part requires a mass in order to follow the laws of physics for the CryENGINE, such as gravity.

See also

- [Creating a new car XML](#)

Defining a sitting location

Up to this point, we haven't addressed the issue of the player hanging out of the side of the vehicle every time they enter it. This is due to the fact that there is no sitting position defined for the driver position and thus leaves the player in the last position and pose before entering the vehicle. In this recipe, we will fix this problem by creating a new helper within the vehicle XML.

Getting ready

Complete *the Creating a new car XML* recipe. Then open `MyVehicle.xml` in Notepad or an equivalent editor.

How to do it...

1. Between the `<Parts>` and `<Seats>` cells, you will need to insert the following lines into the car XML:

```
<Helpers>
  <Helper name="driver_sit_pos" position="-
0.75,0.25,1.5"direction="0,1,0" part="body"/>
</Helpers>
```

2. After writing in the `Helpers` cell, we now need to define on the driver seat which helper the player needs to take in order to be positioned in that seat.
3. For **the** `<Seat name="driver">` cell, **you will need to change the following** property:

```
sitHelper=" "
```

4. Change this value to:

```
sitHelper="driver_sit_pos"
```

5. Save the XML and reload the car in the level.

How it works...

This simple seat helper lets the vehicle code know where the passengers of that particular seat need to be positioned in the vehicle. Be aware that this only aligns the root pivot of the character to this position. So in order to pose a character to sit in this position, it is a good practice to create a new animation graph and animation that moves the character so that the root pivot aligns to a bone on the character such as the pelvis.

There's more...

You can use the HMMWV animation graph to give a seated position to your character. If no changes to the player model have been made to the skeleton, then we will be able to utilize the animation graph from the SDK's HMMWV to pose our character. Within the same `<Seat name="driver">` cell, add the following property:

```
agVehicleName="HMMWV"
```

This will use the animation graph to play the same animations used for entering and exiting the vehicle as well as utilizing the same sitting pose as the one used for the HMMWV.



See also

- ▶ *Creating a new car XML*
- ▶ *Setting up multiple cameras for the car*

Setting up multiple cameras for the car

In this recipe, we will demonstrate how you can set up different cameras at different positions to be utilized within the game. We will demonstrate some of the basic views such as *First Person* and *Third Person* as well as the popular *Wheel rim* camera usually used in action replays in various other media.

Getting ready

Complete the *Creating a new car XML* recipe. Then open `MyVehicle.xml` in Notepad or an equivalent editor.

How to do it...

First Person Camera:

1. After the *Creating a new car XML* recipe, we should already have the majority of what we need in the `<Seats>` cells. For our first camera, we will utilize the `<View class="FirstPerson">` cell to make a new first person camera that we can also look around with.
2. First, we will need a new *Helper*. Copy the following into the `<Helpers>` cells:


```
<Helper name="driver_view" position="-0.75,0.35,2.0"
direction="0,1,0" part="body"/>
```

3. Now, define this `driver_view` helper in the empty `FirstPerson.helper` property:

```
<FirstPerson helper="driver_view" />
```

4. To allow the player to look around in this *First Person* view, we will need to add/change the properties in the `FirstPerson` class to match the following line:

```
<View class="FirstPerson" canRotate="1" rotationMin="-45,0,-170"
rotationMax="45,0,170">
```



The `rotationMin` and `rotationMax` are measured in degrees.

5. You may need to tweak the position a bit, but the result should look something like the following screenshot:



By default, *F1* toggles the player's camera. This also works within vehicles.

Third Person Camera:

Since the creation of the car, the *Third Person* camera is basically set up. However, we may want to adjust some of the properties to the following:

```
<ThirdPerson distance="3" heightOffset="2.5" speed="1" />
```

This will bring the overall distance a little closer, push up the camera's height as well as slow down the camera to induce a little bit of *lag*, allowing the car to move further ahead of the camera a bit when driving really fast or lag a bit on sharp turns.



Wheel Camera:

1. This is a bonus camera that may add a bit of fun when cycling between cameras. To make this camera, we will need to first create a new view. Copy the following into your `<Views>` cell:

```
<View class="FirstPerson" canRotate="0">
  <FirstPerson helper="wheel_cam" />
</View>
```

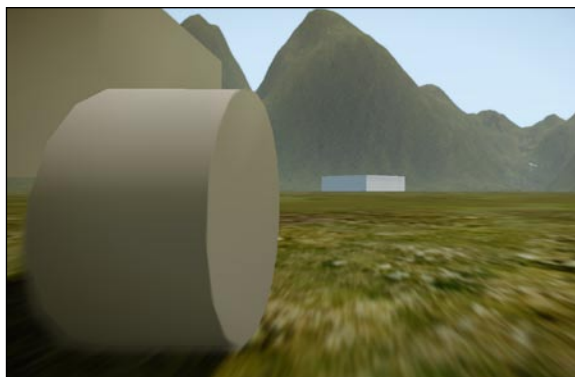
2. Next, we need to create a new helper for this camera:

```
<Helper name="wheel_cam" position="1.9,-0.5,0.5" direction="0,1,0"
part="body"/>
```



This position will depend on the body of the vehicle and the position of the wheels from the mesh.

3. In the end, the desired affect is a *Wheels Camera* that resembles something like the following screenshot:



How it works...

Just like defining a helper position for the sitting position, helpers can be used in a number of different places on the XML and the cameras are no different. Utilizing the positions and directions of the helpers, you will be able to create some interesting camera positions of your own.

There's more...

Some additional cameras such as *SteerThirdPerson* and *ActionThirdPerson* (no longer supported) can also be utilized as other styles of cameras when the player is sitting in the seat of the vehicle. You can find their properties from `def_vehicle.xml` (MyGameFolder\Scripts\Entities\Vehicles).

See also

- ▶ [Creating a new car XML](#)

Need for a machine gun

In this recipe, we will do something fun with the car and attach a machine gun to the vehicle and have it be fired from the driver's seat.

Getting ready

Complete the *Creating a new car XML* recipe. Then **open** `MyVehicle.xml` in **Notepad** or an equivalent editor. Delete the following cells from `MyVehicle.xml`:

```
<Weapons>
  <Primary/>
</Weapons>
<ActionParts/>
```

How to do it...

In the cells that were just deleted (as they were obsolete), copy the following lines:

```
<SeatActions>
  <SeatAction class="Weapons">
    <Weapons isSecondary="0">
      <Weapons>
        <Weapon class="AsianCoaxialGun" part="body">
          <Helpers>
```

```

        <Helper value="mgun_out"/>
    </Helpers>
</Weapon>
</Weapons>
</Weapons>
</SeatAction>
</SeatActions>

```

How it works...

The newly added `<SeatAction>` tag is used for any weapons that are made for the vehicles. The weapon itself can use a helper to fire from and if the weapon is set up on a part, then the weapon will rotate with that part as well (such as a turret).

The flag of `isSecondary` handles whether the weapon is fired from the primary fire button or if it uses the secondary fire button.

See also

- [Creating a new car XML](#)

Giving the car a weak spot

From the *Creating a new car XML* recipe, we skipped the *Damages and Components* section of the XML. Now it is time to return to that section and give this car a weak spot.

Getting ready

Complete the *Creating a new car XML* recipe. Then open `MyVehicle.xml` in Notepad or an equivalent editor.

How to do it...

1. We need to first create a new cell in between `<Physics>` and `<Components/>`. Copy the following code snippet to that location:

```

<Damages submergedRatioMax="0.6" submergedDamageMult="1"
collDamageThreshold="10" groundCollisionMinMult="1.3"
groundCollisionMaxMult="1.4" groundCollisionMinSpeed="8"
groundCollisionMaxSpeed="22">
    <DamageMultipliers>
        <DamageMultiplier damageType="bullet" multiplier="0.125"/>
        <DamageMultiplier damageType="collision" multiplier="1"/>
    </DamageMultipliers>

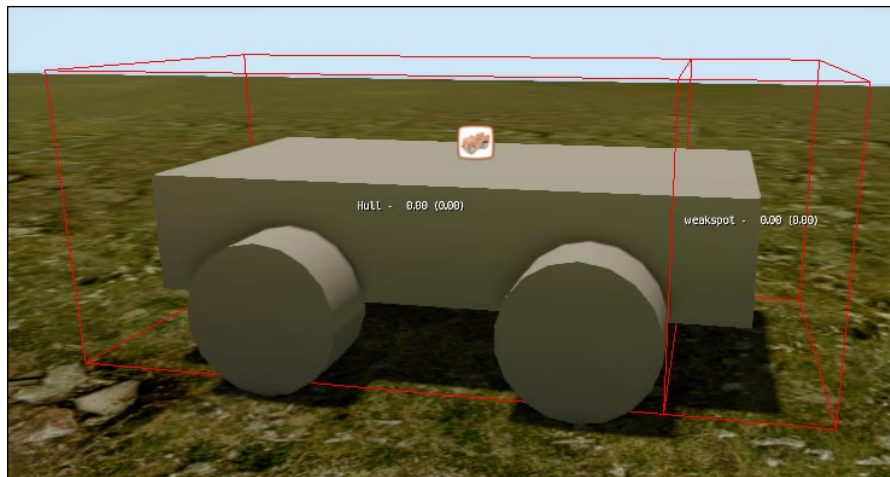
```

```
<DamagesGroups>
  <DamagesGroup name="Destroy" useTemplate="CarDestroy">
  </DamagesGroup>
</DamagesGroups>
</Damages>
```

2. The `<Damages>` tag here handles the global damage done to the entire vehicle, such as if the vehicle is submerged under water, collisions with the entire vehicle, global damage multipliers (which will be further explained in the *How it works...* section) and also the `<DamagesGroups>` tag, which usually handles the destruction of the vehicle. We will cover that later on.
3. Next, we will need to create the components of the vehicle:

```
<Components>
  <Component name="Hull" damageMax="100" position="0,0.5,1"
size="2.5,4,2" useBoundsFromParts="0">
    <DamageBehaviors>
      <DamageBehavior class="Group" damageRatioMin="1">
        <Group name="Destroy"/>
      </DamageBehavior>
    </DamageBehaviors>
  </Component>
  <Component name="weakspot" damageMax="100" position="0,-2,1"
size="2.5,1,2" useBoundsFromParts="0">
    <DamageMultipliers>
      <DamageMultiplier damageType="bullet" multiplier="1"/>
    </DamageMultipliers>
    <DamageBehaviors>
      <DamageBehavior class="Group" damageRatioMin="1">
        <Group name="Destroy"/>
      </DamageBehavior>
    </DamageBehaviors>
  </Component>
</Components>
```

4. The two components set previously create two different bounding boxes. One is the main hull, which wraps around 85 percent of the vehicle at the front, while the other is the weakspot, which covers the back.



How it works...

Since the components do not overlap each other, this allows for the vehicle to respond differently when the vehicle gets hit in either location. In this example, we have set up a *DamageMultiplier* on the *weakspot* component to be much higher for bullets. This will make sure the bullet damage it receives is not reduced and will quickly reach the component's maximum health limit of 100. When that happens it will call the damage behaviour group *Destroy*, which was written at the beginning of the recipe.

The damage group *Destroy* uses the *DamagesTemplate* called *CarDestroy*. This template is a global *DamageBehaviour* that's written in an external XML and can be accessed from other vehicle XMLs for ease of setup.



The `DefaultVehicleDamages.xml` can be found in the following location:
`MyGame\Scripts\Entities\Vehicles\DamagesTemplates\.`

There's more...

Additional damage behaviors can be utilized such as *Burn*, *Effect* (particle effects), *Explosion* (with impulse), *HitPassenger*, and many more. See `def_vehicle.xml` for the complete list.

See also

- *Creating a new car XML*

9

Game Logic

In this chapter, we will cover:

- ▶ How to beam the player to a tag point from a trigger
- ▶ Making the AI go to a location when the player enters a proximity trigger
- ▶ Debugging the Flow Graph
- ▶ Creating a kill counter
- ▶ Rewarding the player for reaching a kill goal
- ▶ Displaying the player's health through a Flow Graph
- ▶ Changing the player camera through key input
- ▶ Creating a countdown timer
- ▶ Creating a sound event

Introduction

While we have explored much of the external scripting in the previous chapters, we shall now look at more of the inner workings of the game logic that is made via a Flow Graph. Being one of the most powerful tools for any designer, in this chapter, we shall explore some of the examples that can be created with the Flow Graph.

How to beam the player to a tag point from a trigger

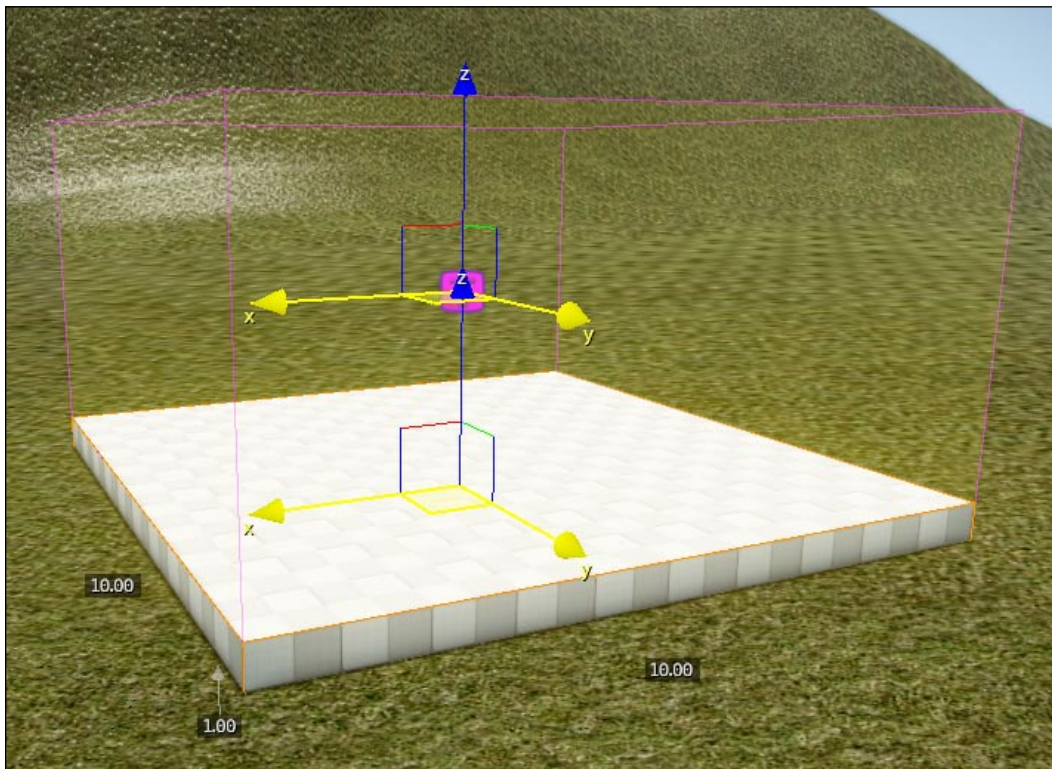
In this recipe, we will demonstrate a basic use of setting up a trigger area and then teleporting the player to a particular location once the player walks inside that trigger area.

Getting ready

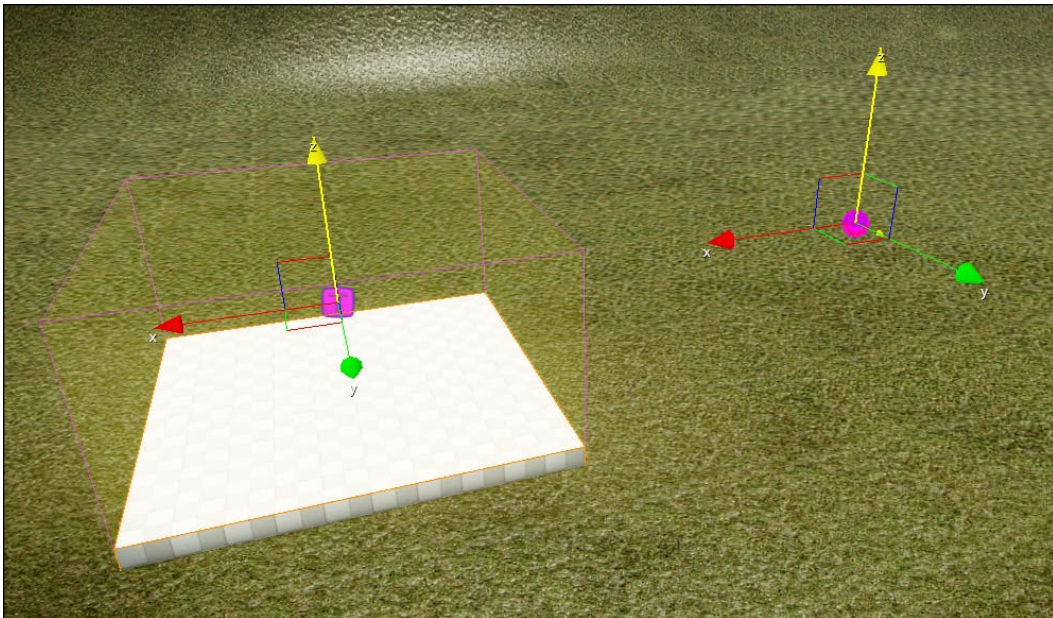
- ▶ Before we begin, you must have the Sandbox open
- ▶ Then open `My_Level.cry`

How to do it...


1. In the **RollupBar**, click on the **Entities** button.
2. Under the **Triggers** section, select **ProximityTrigger**.
3. Place this **ProximityTrigger** on the ground with the following property changes:
 - ❑ **DimX = 10**
 - ❑ **DimY = 10**
 - ❑ **DimZ = 5**
 - ❑ **OnlyPlayer = True**
4. Next, draw down a solid to mark the area where we might see the trigger area that the player needs to enter. Match the dimensions to the following—**10x10x1** meter.



5. Next, place down a tag point to reference as the beaming destination.
6. In the **RollupBar**, click on the **AI** button.
7. From the **Object Type**, select **Tagpoint**.
8. Place the tag point several meters away from the trigger.



9. And finally, we need to create the logic within the Flow Graph to trigger the beaming event for the player.
10. Right-click the **ProximityTrigger** | **Create Flow Graph**.
11. Name the Flow Graph `Beam_Trigger` and click **OK**.
12. Go back into the editor's perspective viewport and select the **ProximityTrigger**.
13. Open the Flow Graph editor again and select the **ProximityTrigger** within the FG hierarchy.

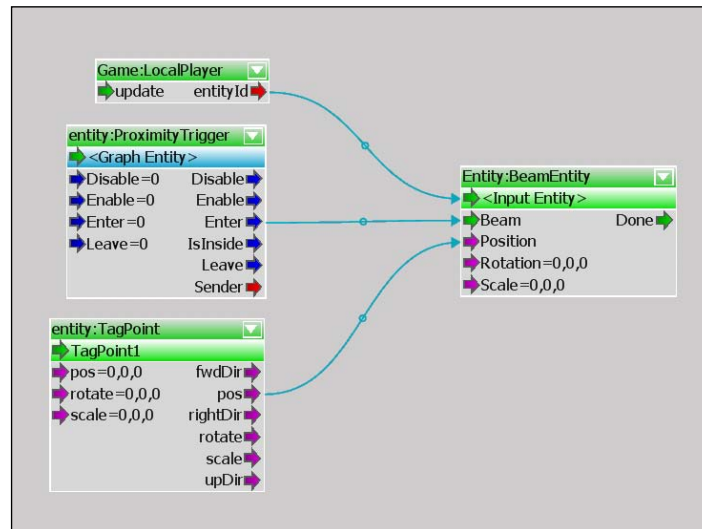

 You can re-open the Flow Graph from the main
Tool Bar | View | Open View Pane | Flow Graph.

14. Right-click in an open space and click **Add Selected Entity**.
15. Repeat steps 3 to 5 for the tag point.



You can also select multiple entities and add them all at the same time within the Flow Graph.

16. Right-click **Add Node | Entity | Beam Entity**.
17. Then right-click **Add Node | Game | Local Player**.
18. Link the **Enter** output from the **ProximityTrigger** to the input of **Beam** on the **BeamEntity** node.
19. Link the **entityId** of the **LocalPlayer** node to the input of **Choose Entity** in the **BeamEntity** node.
20. Link the **pos** output from the **Tagpoint** node to the **Position** input on the **BeamEntity** node.



Now every time the player walks into the defined proxy trigger area, the player will be beamed to the tag point's world coordinates.

How it works...

Beaming within CryENGINE 3 is like instantly teleporting an entity to any defined location in the level. With this setup, we are telling the code to beam the player to the tag point's defined Vec3 position within the world as soon as they enter the trigger. This works with any entity type within CryENGINE 3.

See also

- ▶ The *Debugging the Flow Graph* recipe

Making the AI go to a location when the player enters a proximity trigger

In this recipe, we will demonstrate how you may be able to give a basic `Goto` command to an **AI (Artificial Intelligence)** when the player enters a trigger.

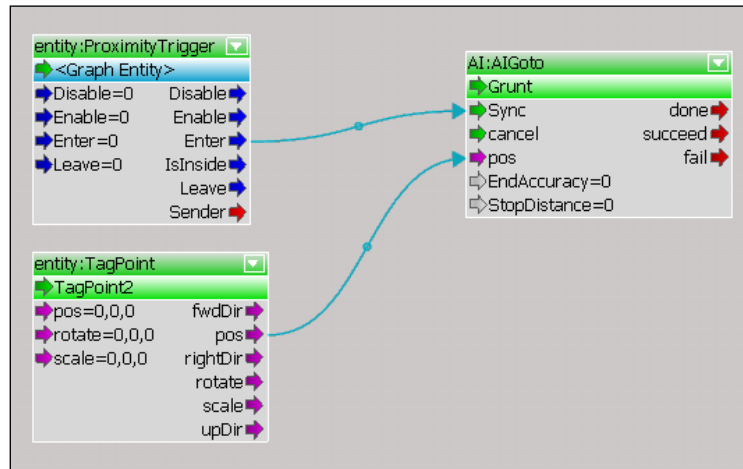
Getting ready

- ▶ Open Sandbox and then open `My_Level.cry`
- ▶ Complete the *How to beam the player to a tag point from a trigger* recipe to learn how to place the proximity triggers
- ▶ Then place down a new Grunt

How to do it...

1. First, place down a proximity trigger and a solid with the same dimensions and properties as the previous recipe.
2. Then place down an AI tag location within that same trigger area.
3. Next, create a new Flow Graph from the trigger area called `AI_Goto`.
4. Within the Flow Graph, create the following logic:
 - ❑ From the perspective viewport of the editor, select the **ProximityTrigger** and the **TagPoint**
 - ❑ Add **Selected Entities** to the **Proximity Trigger** Flow Graph
 - ❑ Add **Node | AI | AIGoto**
5. Select a Grunt from the perspective viewport.
6. Back in the Flow Graph, right-click **AIGoto | Assign Selected Entity**.
7. Link **Enter** output from the **ProximityTrigger** to the **Sync** input of **AIGoto**.

8. Link **pos** output from the **TagPoint** to the **pos** input of **AI:Goto**.



With that logic now set up, the AI should now attempt to go to that tag point (unless its behavior is interrupted by perceiving the player).

How it works...

With the AI:Goto node, this trigger is a nudge to the AI's behavior tree to signal a basic command for the AI to go to a specific location. Unless the AI's behaviour is not currently busy with other tasks, the AI will proceed to the defined location.

There's more...

Here is an extra Flow Graph node you might want to try as well.

AI:GotoEx

An extended version of the AI:Goto node is the AI:GotoEx. This node allows the designer to forcefully command the AI to go to a particular location regardless of the current condition of its state. This is a key node to use if the designer wants to force the AI to retreat for example.

See also

- The *Debugging the Flow Graph* recipe

Debugging the Flow Graph

Now that we have some basic Flow Graphs already made, we will now look at how to debug them to find out when certain events are firing. This will aid any designer to track down any Flow Graph logic errors that might occur.

Getting ready

- ▶ You should have completed the *How to beam the player to a tag point from a trigger* recipe
- ▶ Open `My_Level.cry` within the Sandbox

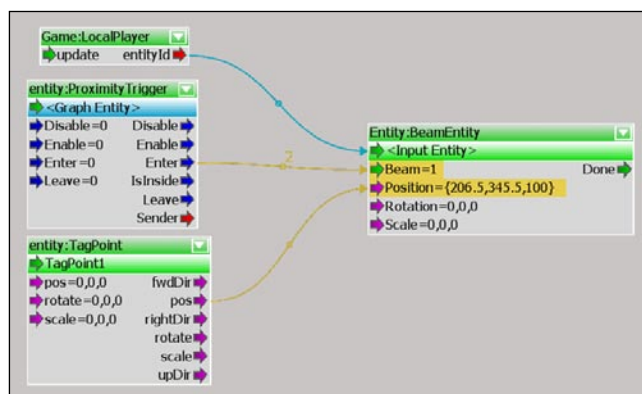
How to do it...

Open up the Flow Graph editor and open the Flow Graph. To enable the Flow Graph debugging, click on the icon that resembles a bug:



How it works...

Each time you enter the game mode with the debugger turned on, any of the Flow Graph outputs that are triggered within the level will change their arrows from blue to yellow. They will also display numbers above the yellow arrow if the trigger has been triggered more than once, which is equal to the number of times the output has fired.



There's more...

To clear the Flow Graph debug results, click on the *Trash Icon* next to the *Debug Icon* within the Flow Graph:



See also

- ▶ The *How to beam the player to a tag point from a trigger* recipe

Creating a kill counter

In this recipe, we will look at one of the many methods to make a quick kill counter for the player based on the *HitInfo* taken from *raycasted* weapons (melee, bullets, and so on) and also using the Grunt entity types as the targets.

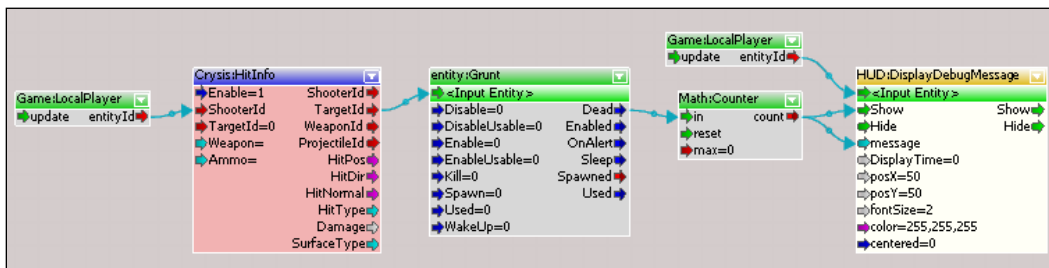
Getting ready

- ▶ Open `My_Level.cry` within the Sandbox

How to do it...

1. Begin by placing an **AreaTrigger** entity onto the map to use as the container for the Flow Graph.
2. In the **RollupBar**, click on the **Entities** button.
3. Under the **Triggers** section, select **AreaTrigger**.
4. Right-click the newly placed **AreaTrigger** and create a new Flow Graph. Name the Flow Graph as `KillCounter`.
5. With the Flow Graph open and the new **AreaTrigger** selected, add in the following nodes:
 - ❑ **2x Game:LocalPlayer**
 - ❑ **Crysis:HitInfo**
 - ❑ **Math:Counter**
 - ❑ **HUD:DisplayDebugMessage**
6. Set **Crysis:HitInfo Enable** to **true (1)**.

7. You will also need to place down at least one Grunt within the level. Then with the Grunt selected, go back into the **AreaTrigger** Flow Graph and right-click **Add Selected Entity**.
8. Link the Flow Graph together as follows:
 - ❑ **Game:LocalPlayer entityId** out to **Crysis:HitInfo ShooterId** in.
 - ❑ **Crysis:HitInfo TargetId** out to **entity:Grunt Grunt #** in. (This will change to **<Input Entity>** as these nodes take the entity IDs.)
 - ❑ **entity:Grunt Dead** out to **Math:Counter in** in.
 - ❑ **Math:Counter count** out to **HUD:DisplayDebugMessage Show** in AND **message** in.
 - ❑ Second **Game:LocalPlayer entityId** out to **HUD:DisplayDebugMessage Choose Entity** in.
9. The resulting Flow Graph should look like the following:



How it works...

Using the **Crysis:HitInfo** node within the Flow Graph checks to see what the player is hitting with any *Raycasting* weapons such as bullets or melee for example. Each time the player lands a hit on an entity, the *TargetId* is then triggered in this node. As we are comparing against Grunt type entities, it checks to see if that *TargetId* died. If the Grunt has died from this event, it forwards a true value in the Boolean only once, which gets counted by the **Math:Counter**. The total value of the *Math Counter* is then displayed back to the player showing how many times the player has killed the Grunts.

The number displayed will appear in the upper-left corner of the screen after the successful kills have been made:



See also

- ▶ The *Rewarding the player for reaching a kill goal* recipe

Rewarding the player for reaching a kill goal

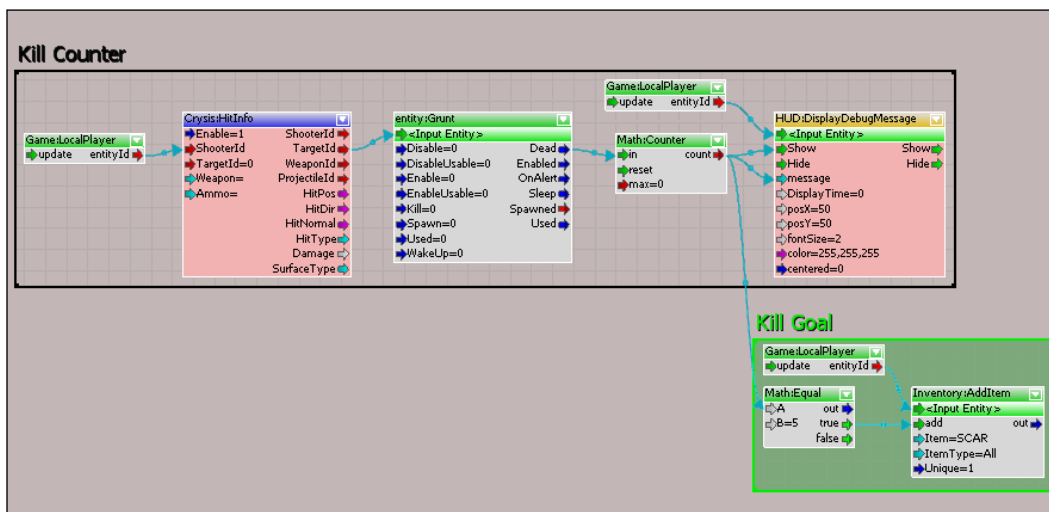
This recipe is a slight extension to the previous one. There are plenty of ways that the designers may want to reward the player in their title, but in this recipe, we will look at getting the player to kill five Grunts using just melee and then reward them by giving them a SCAR assault rifle.


Getting ready

- ▶ You should have completed the *Creating a kill counter* recipe
- ▶ Then open `My_Level.cry` within the Sandbox

How to do it...

- Open the *KillCounter AreaTrigger* Flow Graph and add in the following flow nodes:
 - Game:LocalPlayer**
 - Math:Equal**
 - Inventory:AddItem**
- Set **Math:Equal B** to **5**.
- Link the Flow Graph together as follows:
 - Math:Counter count** out to **Math:Equal A** in
 - Math:Equal true** out to **Inventory:AddItem add** in
 - Game:LocalPlayer entityId** out to **Inventory:AddItem Choose Entity** in
- The end resulting Flow Graph should look like the following:




 The *Kill Counter* and *Kill Goal* colored boxes around the nodes are comment boxes. To add a comment box, right-click in the Flow Graph and click **Add Comment Box**.

How it works...

Branching off the **Kill Counter**, we can use the **Math:Counter** to find what kill number the player is at. Using the **Math:Equal** node, we can run a simple comparison to any value we please and in this example we used **5**. So as soon as the **Math:Counter** reaches a value of **5**, the **Math:Equal** node outputs a true statement and allows the SCAR to be added to the player's inventory.

See also

- ▶ The *Creating a kill counter* recipe

Displaying the player's health through a Flow Graph

Unfortunately, with the **CryENGINE 3 SDK** there is no current HUD element in place that displays the player's health out of the box. In this recipe, however, we will look at adding in a health display through the Flow Graph that will show the player's health in a numerical fashion.

Getting ready

Open `My_Level.cry` within the Sandbox

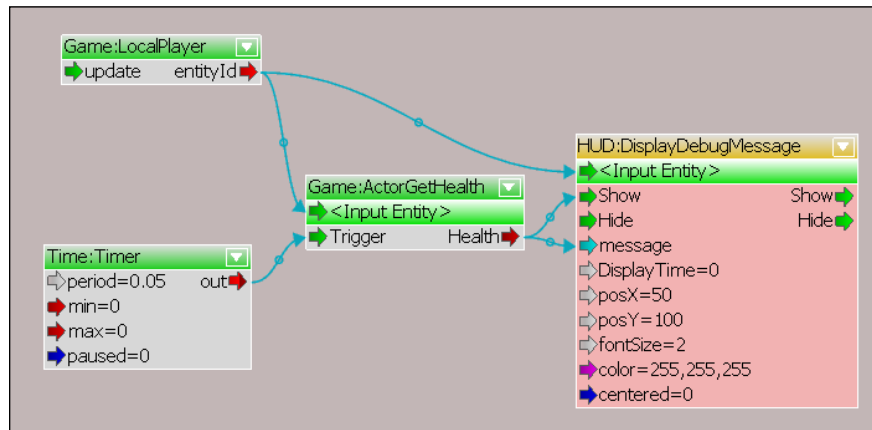
How to do it...

1. Begin by placing an **AreaTrigger** entity onto the map to use as the container for the Flow Graph.
2. In the **RollupBar**, click on the **Entities** button.
3. Under the **Triggers** section, select **AreaTrigger**.
4. Right-click the newly placed **AreaTrigger** and create a new Flow Graph. Name the Flow Graph `PlayerHealth`.
5. With the Flow Graph open and the new **AreaTrigger** selected, add in the following nodes:
 - ❑ **Game:LocalPlayer**
 - ❑ **Time:Timer**
 - ❑ **Game:ActorGetHealth**
 - ❑ **HUD:DisplayDebugMessage**
6. Set **Time:Timer** period to **0.05**.



You may also need to adjust the *posX* or *posY* on the **HUD:DisplayDebugMessage** if you have the **Kill Counter** as well.

7. Link the Flow Graph together as follows:
 - ❑ **Game:LocalPlayer entityId** out to **Game:ActorGetHealth Choose Entity** in
 - ❑ **Time:Timer out** out to **Game:ActorGetHealth Trigger** in
 - ❑ **Game:ActorGetHealth Health** out to **HUD_DisplayDebugMessage Show AND message** in
 - ❑ **Game:LocalPlayer entityId** out to **HUD_DisplayDebugMessage Choose Entity** in
8. The resulting Flow Graph should look like the following:



We should now see a numerical value of **100** in the upper-left corner of the screen the next time we enter into Game Mode on this level.

How it works...

Although this is kind of an expensive method of measuring the player's health, the timer is periodically getting the player's health (every 0.05 seconds) and outputting it to a HUD message for the player to read.

Changing the player camera through key input

In this recipe, we will look at a simple setup to allow the player to switch from the First Person view to the Third Person view.

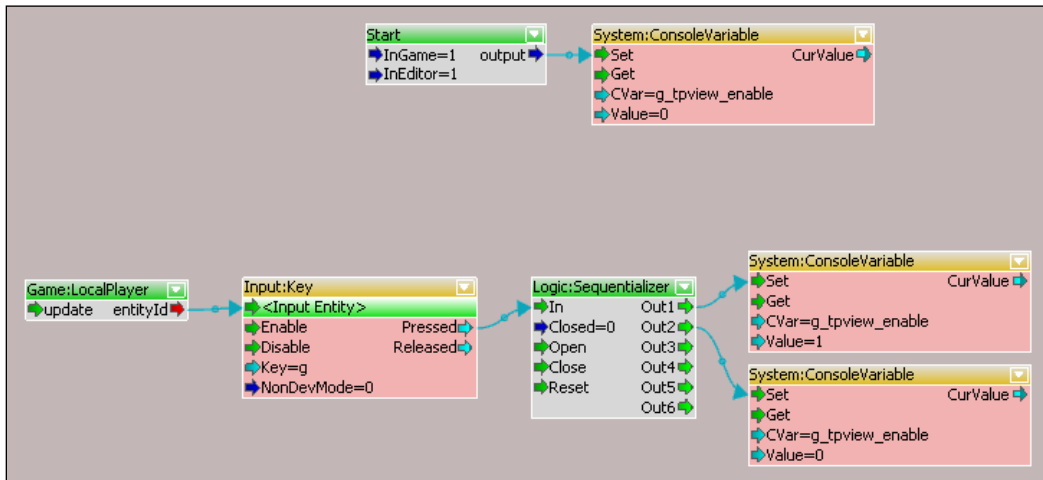
Getting ready

Open `My_Level.cry` within the Sandbox

How to do it...

1. Begin by placing an **AreaTrigger** entity onto the map to use as the container for the Flow Graph.
2. In the **RollupBar**, click on the **Entities** button.
3. Under the **Triggers** section, select the **AreaTrigger**.
4. Right-click the newly placed **AreaTrigger** and create a new Flow Graph. Name the Flow Graph as `PlayerViewToggle`.
5. With the Flow Graph open and the new **AreaTrigger** selected, add in the following nodes:
 - **Game:LocalPlayer**
 - **Input:Key**
 - **Logic:Sequentializer**
 - **3x System:ConsoleVariable**
 - **Misc:Start**
6. Set the input **Key** = `g`.
7. Set all three **System:ConsoleVariable Cvar=g_tpview_enable**.
8. Set one **System:ConsoleVariable Value=1**.
9. Link the Flow Graph together as follows:
 - **Game:LocalPlayer entityId** out to **Input:Key Choose Entity** in
 - **Input:Key Pressed** out to **Logic:Sequentializer In** in
 - **Logic:Sequentializer Out1** to **System:ConsoleVariable Set** in (this needs to be `g_tpview_enable = 1`)
 - **Logic:Sequentializer Out2** to **System:ConsoleVariable Set** in (this needs to be `g_tpview_enable = 0`)
 - **Misc:Start output** out to **System:ConsoleVariable Set** in (this needs to be `g_tpview_enable = 0`)

10. The resulting Flow Graph should look like the following:



How it works...

This Flow Graph has an input listener that waits for the specific key to be pressed by the **entityId** it is assigned (in this case, the player). Once this key is pressed, it will toggle the console variable that enables Third Person view. The **Misc:Start** to force set the **CVar** is a preventative measure to make sure the player is in First Person on Game Start.

This is not an ideal method of modifying the view of the player for final release as this should actually be done in code. However, this Flow Graph will do the job until code can perform this same function.

There's more...

An alternative camera to the standard Third Person view is the `goc_camera`, which – by default – hangs over the right shoulder of the player. Explore the `goc_camera`'s capabilities with the following CVar:

- ▶ `goc_enable`
- ▶ `goc_targetx`
- ▶ `goc_targety`
- ▶ `goc_targetz`
- ▶ `g_tpview_force_goc`

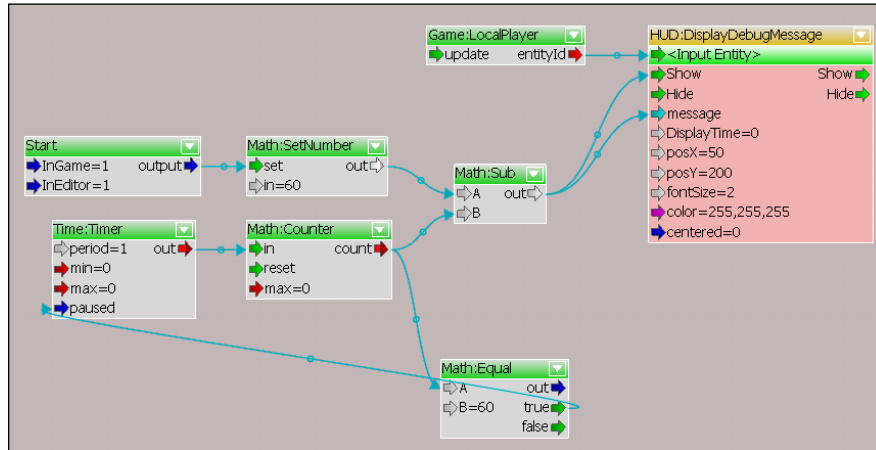
Creating a countdown timer

In this recipe, we will look at one possible way to create a countdown timer from 60 to 0 seconds that the player will see on-screen.

How to do it...

1. Begin by placing an **AreaTrigger** entity onto the map to use as the container for the Flow Graph.
2. In the **RollupBar**, click on the **Entities** button.
3. Under the **Triggers** section, select the **AreaTrigger**.
4. Right-click the newly placed **AreaTrigger** and create a new Flow Graph. Name the Flow Graph as `CountDownTimer`.
5. With the Flow Graph open and the new **AreaTrigger** selected, add in the following nodes:
 - ❑ **Game:LocalPlayer**
 - ❑ **HUD:DisplayDebugMessage**
 - ❑ **Misc:Start**
 - ❑ **Math:SetNumber**
 - ❑ **Math:Sub**
 - ❑ **Time:Timer**
 - ❑ **Math:Counter**
 - ❑ **Math:Equal**
6. Set **Time:Timer period=1**.
7. Set **Math:SetNumber in=60**.
8. Set **Math:Equal B=60**.
9. Link the Flow Graph together as follows:
 - ❑ **Game:LocalPlayer entityId** out to **HUD:DisplayDebugMessage Choose Entity** in
 - ❑ **Misc:Start output** out to **Math:SetNumber set** in
 - ❑ **Math:SetNumber out** out to **Math:Sub A** in
 - ❑ **Time:Timer out** out to **Math:Counter in** in
 - ❑ **Math:Counter count** out to **Math:Sub B** in
 - ❑ **Math:Counter count** out to **Math:Equal A** in
 - ❑ **Math:Equal true** out to **Time:Timer paused** in
 - ❑ **Math:Sub out** out to **HUD:DisplayDebugMessage Show AND message** in

10. The resulting Flow Graph should look like the following:



How it works...

A simple use of the timer and math counter outputs the value that is used to subtract from the 60 value from **Math:SetNumber**. As soon as the counter reaches 60, the **Math:Equal** node sends a signal to the timer to pause itself and stop the count at 0.

There's more...

One addition to this could be the node of **Math:Round**. Linking the result of the **Math:Sub** to the **In** input of this node and linking the **outRounded** value to the **HUD:DisplayDebugMessage** will round the numbers down to an integer.

10

Track View and Cut-Scenes

In this chapter, we will cover:

- ▶ Creating a new Track View sequence
- ▶ Animating a camera in the Track View
- ▶ Triggering a sequence using the Flow Graph
- ▶ Animating entities in the Track View
- ▶ Playing animations on the entities in the Track View
- ▶ Using console variables (CVars) in the Track View
- ▶ Using track events

Introduction

The Track View editor is the embedded Sandbox cut-scene creation tool for making interactive movies, such as sequences with timeline dependent control over objects and events in CryENGINE 3. For those already familiar with the CryENGINE 2 Track View system, you will see that it is quite similar, although improvements have been made to tools such as curve editor and director tracks can now be used for what used to be complex sequences.

Using Track View, creating cinematic cut-scenes and scripted events are both possible, allowing you to sequence objects, animations, and sounds in a scene that can be triggered during a game and played either as a detached cut-scene from the third person perspective, or from the first person perspective of the player as he plays the game.

Track View and Cut-Scenes

Sequences created with the Track View can be triggered in a game with a specific Flow Graph node. Different properties enable sequences to range from passive in-game scenarios up to fully uncoupled cut-scenes.



This system will be familiar to anyone who has used animation software such as 3ds Max, but this guide will also help those unfamiliar with cut-scene editors to start creating simple scenes for your levels.

Creating a new Track View sequence

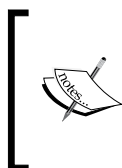
Creating Track View sequences are very akin to creating animation within the key frame animation DCC tools.

This recipe will familiarize you with the Track View editor and take you through some of the important interface controls.

How to do it...

Let's create our own Track View sequence:

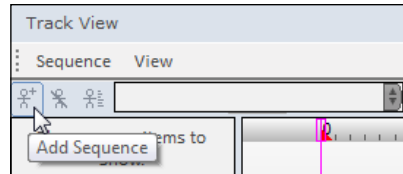
1. Open Sandbox and then open any level.



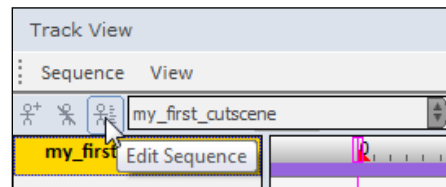
Any time you add cut-scenes or Track View sequences to a level, you should always make a new layer to add your objects onto. This will allow you to control all these objects independently of the level and hide or show them when needed.

An object is placed onto whatever layer is active when the new Track View sequence is created.

2. Add a layer called `Cinematics`.
Once the layer is added, ensure that it is the active layer by checking that it is highlighted.
3. To create a **Track View sequence**, open the **Track View** panel selecting the menu option **View | Open View Pane | Track View**.
4. Click on the **Add Sequence** icon in the **Track View** editor to create a new sequence:



5. Clicking the **Add Sequence** icon will present you with the opportunity to name the sequence. Name this sequence `my_first_cutscene`.
6. When the sequence is added, a new object is placed on the active layer. This object is called a sequence object and will store all the information for our Track View sequence.
7. Now that we have created a new sequence, it's time to adjust some of the fundamental settings for it.
8. To do this, click the **Edit Sequence** icon:

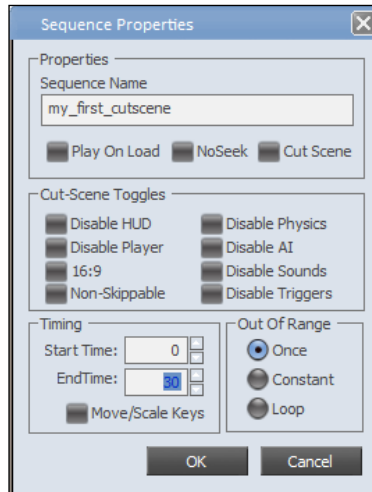


Upon clicking **Edit Sequence**, you will be presented with the properties window. Many of the properties here are self-explanatory.

9. In our case, let's simply adjust the length of the sequence by changing the **EndTime** parameter to **30**.

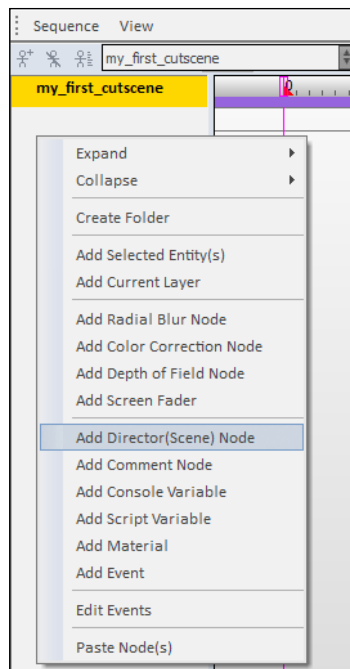


By default, the Track View editor's timeline will display time in seconds.



Now that we have the basic setup, let's add some tracks to our sequence:

1. The first track that we will add will be a director node.
2. The **director node** allows us to activate different cameras throughout the course of the Track View allowing a simple form of camera cutting. In our example, we will only use a single camera but it is a good practice to always add a director node to your sequences.
3. To add the director node, right-click in the empty whitespace in the left-hand side of the Track View editor. You will see a large dialog with many different track additions that you can make.
4. Select to add a director (scene) node:



We will use this director node to activate different cameras in the following recipe.

How it works...

This sequence that we just added is the cut-scene itself. For each sequence, you can add a number of nodes for each object that you use, including the top-level node of the scene itself—this was what we added the director node to. For each node that you have, you can have a number of tracks, depending on what kind of node it is. Each track is measured in seconds, and has points marked on it to indicate where a key frame or event starts.

There's more...

You will likely want to know more about the available properties in the sequence as well as some additional tracks that you can add to the director node.

Available tracks in the director node

The director node has unique tracks that can be added to it. Adding a track to any node can be done simply by right-clicking the node and selecting an available track; some of these include:

- ▶ *Console* allows a key frame to contain information to be passed to the engine's console. An example could be changing the level of detail settings to high for cut-scenes.
- ▶ *Capture* allows a capture command to be sent to the engine to allow for the output of the **Track View** cut-scenes to sequentially capture frames.

Sequence properties

Selecting the **Disable Hud** checkbox disables the default player's HUD. Selecting the **Disable Player** checkbox hides the player during the scene. **16:9** enables the black bar effect. **Non-Skippable** means that if the player decides to skip the scene, it will continue to play.

See also

- ▶ Go to **the Animating a camera in the Track View** recipe in this chapter to start creating your own cut-scenes
- ▶ Go to the *Animating entities in the Track View* later in this chapter to animate objects in your sequence

Animating a camera in the Track View

This recipe will take you through adding a camera to a Track View sequence and animating it. We will create a flythrough of our level using some basic animation techniques and then play it back in the editor.

Getting ready

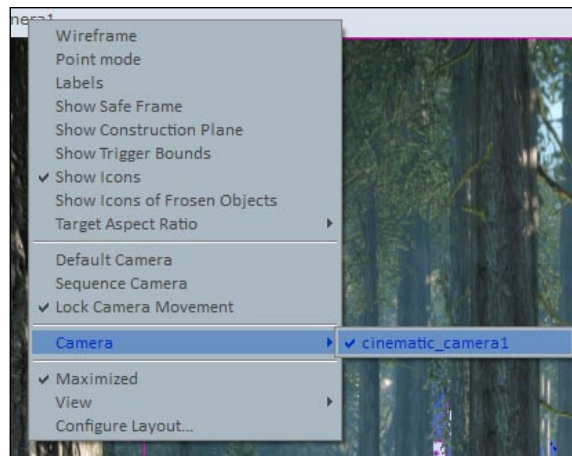
To follow this recipe, you should have created a new **Track View** sequence with a director track already added.

How to do it...

Having added a new track view sequence and a director node, we need some objects to direct:

1. First navigate to the *misc* section in the rollout bar and click-and-drag the camera entity into the level.
2. This will create a wireframe preview of the camera's view. You can move this object in the world as you would with any other entity.

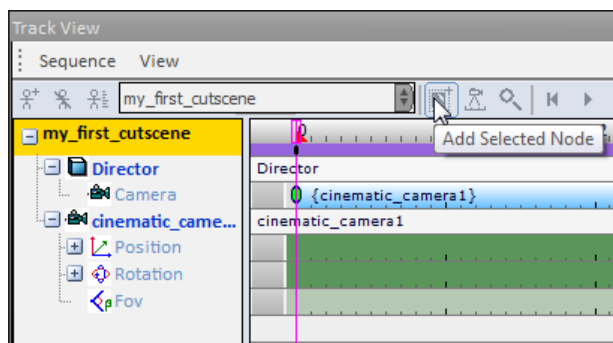
3. Rename this camera to `cinematic_camera1`. At this point, you should adjust your view to be able to quickly preview the actual camera view.
4. You can do this by configuring your viewport layout to have a camera and perspective view concurrently.
5. For this example, however, we will only animate the camera movement from the camera's viewpoint and not from the perspective view.
6. To do this, change your active view to **cinematic_camera1** by right-clicking on the top bar in the viewport:



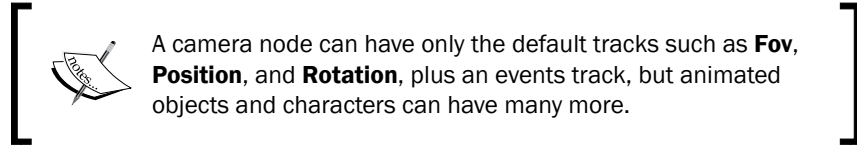
7. Re-open the dialog and also de-activate the **Lock Camera Movement**. This will allow us to control the movement of the camera from its own perspective.

Back in the **Track View** editor, let's now add this camera:

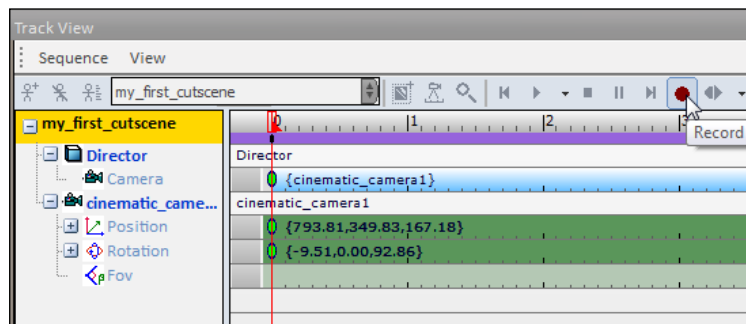
1. To add any entity to a Track View sequence, simply select the entity in the editor, in our case the **cinematic_camera1** entity, and then add it to the track view sequence using the add selected entity icon.



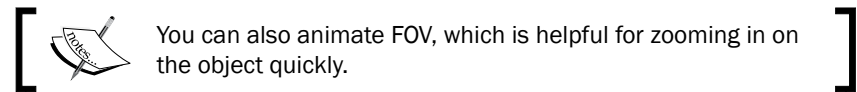
You will notice that when the camera is added to the sequence, it already has some default tracks attached.



2. Having unlocked the camera movement, we must now click the record icon in the Track View to automatically key frame any movement changes to an object in the editor.
3. Select the camera in the editor; you can do this quickly by double-clicking the entity in the Track View. Click **Record** and move the camera to its initial start position.
4. You can move the camera in the exact same way you navigate the viewport in the Sandbox. It does however, allow us to do interesting motions, such as banking or rotating the camera.
5. Navigate the camera to a good starting position where you'd like the flythrough of the environment to begin.



6. Notice that key frames are added for **Position** and **Rotation** automatically while the record icon is active.



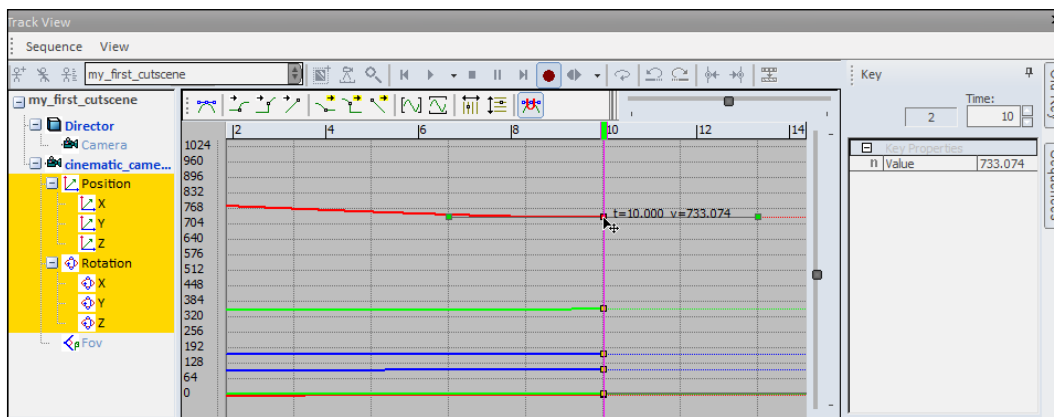
7. Next, move the active time slider to 10 seconds.
8. Keeping in mind that the animation will be interpolated between two key frames, make another key frame at 10 seconds by moving the camera to a different position.

At this new position, let's add some roll to the camera:

1. With the camera selected, click the select and rotate tool and then in the type-in dialogs at the bottom of the view port, type in a value in the **Y** parameter. The higher the value, the more the camera will roll. Use negative values to roll in the other direction.



2. Another effective way of adding variations to the movement is by using the curve editor. The curve editor can be accessed through the **View** section of the Track View editor.
3. Change the view to the curve editor and select and drag one of the axis key frames to a different location at frame **10**. Notice that since the key frames are stored in three XYZ channels, position keys and rotation keys can be edited independently of each other. Position keys are stored in world coordinates and rotation keys are stored in local coordinates. You may also notice the Bezier modifiers on each key frame. This will allow tangents to be adjusted between key frames.



4. Change the view back to the dope sheet and scrub the time slider from frame **10** to frame **0** and back. Notice that you are instantly able to preview the animation!
5. Drag the time slider to 20 seconds and create a new position for the camera in the direction the camera is currently facing forward to the final position for this recipe. Since we are animating a flythrough, you can of course create as many key frames as you like by dragging the time slider forward and moving the camera to a new position for as long as you want your sequence to last.

For this recipe, our sequence is only three key frames long.

6. Preview the entire sequence by clicking the play icon.
Having created key frames at 0, 10, and 20 seconds, you may find that the camera is too fast or too slow. There are two ways to adjust this, first is the most accurate is to select all the key frames at a particular time and bring the time between key frames down by dragging them closer; alternatively, drag them further apart for longer sequences.
7. To return to the default camera view, click the default camera, where you unlocked the camera's movement.
Make sure to save this sequence as we will reuse it later on in this chapter.
8. Save the .cry file as `Forest_sequences.cry`.

How it works...

This sequence that we have created can be triggered via different means during the game mode for a variety of purposes. A simple flythrough can be used to introduce areas, give the player a unique view of events, and to further story and narrative. It is also commonly used to create game play specific events to occur, such as the movement of certain entities within the world and so forth.

There's more...

You may want to know more about the playback speed or some of the other available tracks for the camera.

FOV

The **Field of View (FOV)** on a camera can be set and animated by changing the value on the camera entity, which will be key framed while the record is active. Alternatively, you can type in the values to a manually created key frame by double-clicking anywhere on the FOV track. Some good FOV values for cut-scenes are usually 35-45 depending on the required shot.

Playback speed

You can adjust the play speed of the Track View editor by clicking the pull-down icon beside the play button.

The options available will not translate to in-game triggering of this sequence and will only be used for previewing in the editor.

Curve editor

Animators will likely be familiar with a curve editor approach to key framing objects within Track View. You can adjust your view to contain both the curve editor and dope sheet by selecting both from the view menu.

There are many curve editor tools; most of them pertain to the editing of the tangents between key frames.

See also

- ▶ Go to the *Creating a new Track View sequence* recipe as it is required that you start a new sequence to complete the previous recipe
- ▶ Continue on to the *Triggering a sequence using the Flow Graph* recipe to learn how to trigger your cut-scene

Triggering a sequence using the Flow Graph

Depending on the game or scenario you may be creating, you will likely have different requirements for when and how cut-scenes should be triggered. This is catered quite nicely for designers because of the cross communication within certain tools and systems of the engine. In this recipe, we will use a simple flow graph to trigger a cut-scene when a player walks into a certain trigger.

Getting ready

You must have a level open that has a previously saved Track View sequence in it.

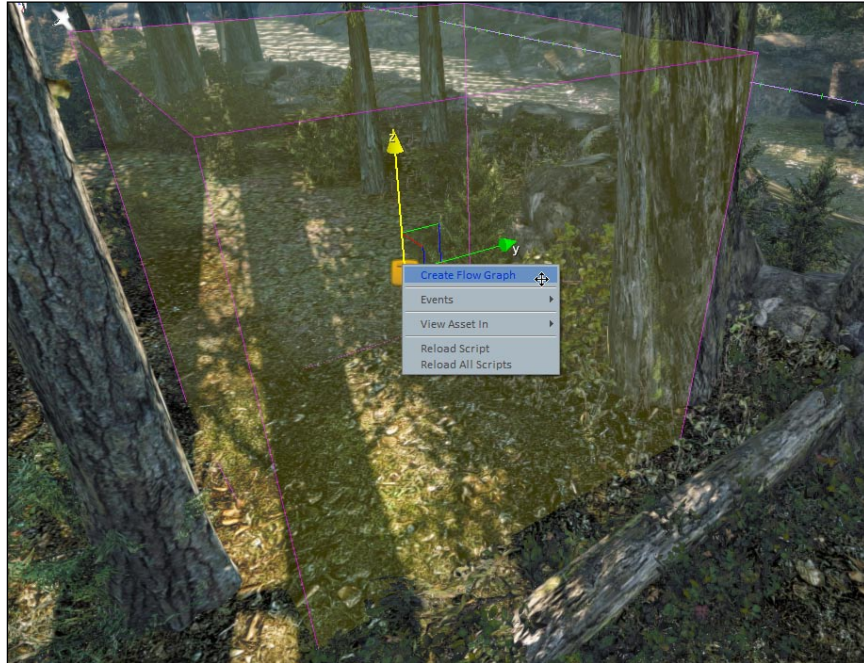
If you have already completed the *Animating a camera in the Track View* and the *Creating a new Track View sequence* recipes then you can use the `my_first_cutscene` sequence.

How to do it...

Let's set up a trigger to launch our cut-scene from game mode:

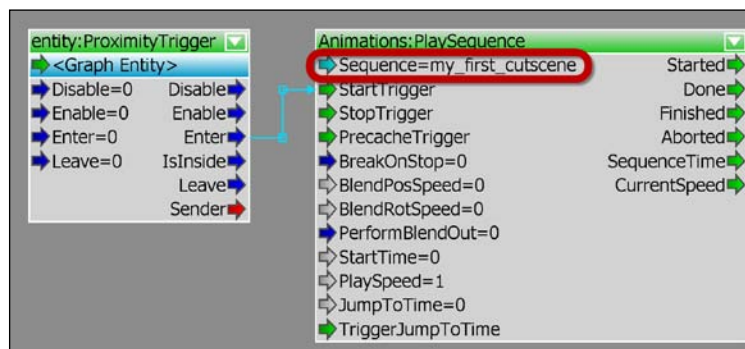
1. Click-and-drag a proximity trigger into the level. This can be found under the entities/triggers section in the Rollout bar.
2. Once you have added it to the level, set the trigger bounds to be big enough to trigger when the player enters it. The values are as follows:
 - ☐ DimX Value="5"
 - ☐ DimY Value="5"
 - ☐ DimZ Value="5"

- Next, right-click the proximity trigger and create a Flow Graph on it:



- Name the Flow Graph `cutscene_trigger`.
- Next, create the following Flow Graph by first adding the proximity trigger itself by right-clicking and selecting **Add selected entity**.
- Also add a the **Animations:PlaySequence** flow node.
- Connect the **Enter** output of the proximity trigger to the start of the trigger of the **PlaySequence** node.

Your resulting flow graph will look similar to the following screenshot:



You can now enter the game mode and whenever you enter the proximity trigger, the sequence is defined in the sequence parameters of the **PlaySequence** flow node.

How it works...

As you can see, one of the fastest ways to trigger a sequence is to attach it to a trigger object such as the proximity trigger, which can be positioned in the level.

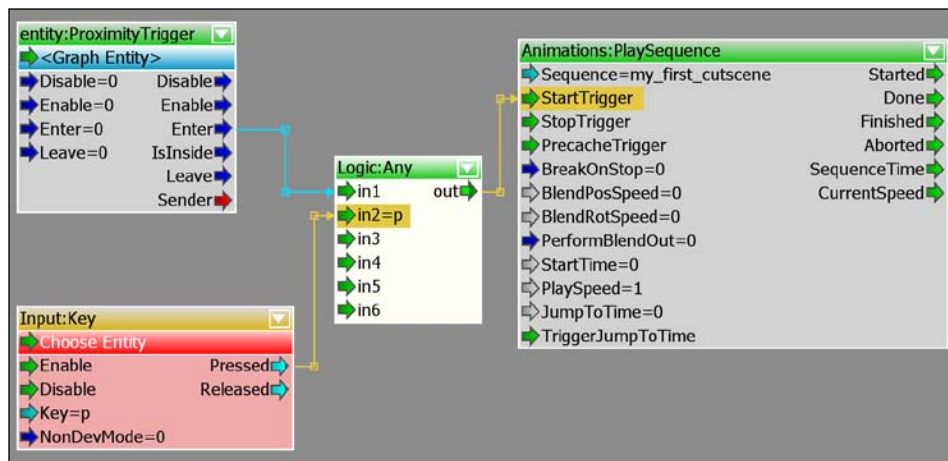
By building a Flow Graph that connects the trigger output of the **ProximityTrigger** to the start trigger of a **PlaySequence** Flow Graph node, one can easily activate a sequence by having a player enter the **ProximityTrigger** in the game.

There's more...

You may want to expand on this Flow Graph or learn how to use an easier testing method for launching cut-scenes in CryENGINE.

Useful debugging trigger

Sometimes for cut-scenes, it can be easier to not have to enter a trigger every time you wish to run the cut-scene. Another useful trigger is the **input:key** flow node that allows you to press a defined key that will trigger the cut-scene.



Start time property

When a sequence is triggered from the **PlaySequence** node, you can force it to jump to a particular start time. This can be useful if you wish to have multiple different events stored within a single sequence.

Break on stop property

When the sequence is stopped, by default, the time slider will go to the very last frame and trigger the logic that may be stored there. This is typically used because when a scene is skipped, you can still trigger all the logic and positional information on that last frame. Setting **Break on Stop** to **true** will overwrite these defaults and cause the logic on the last frame not to be triggered.

See also

- ▶ Refer to the *Debugging the Flow Graph* recipe in *Chapter 9, Game Logic* if you cannot get the Flow Graph to function
- ▶ Continue to the next recipe to animate entities in your cut-scene

Animating entities in the Track View

The animation of objects and entities in the Track View can be as complex or as simple as the animator or the designer requires. In this recipe, we will animate a dead tree falling into water. This will involve animating the tree itself as well as triggering some particle effects at specific times. The interesting part about this will be that we are not making a cut-scene but rather a scripted game play event that will alter the path of the character.

Getting ready

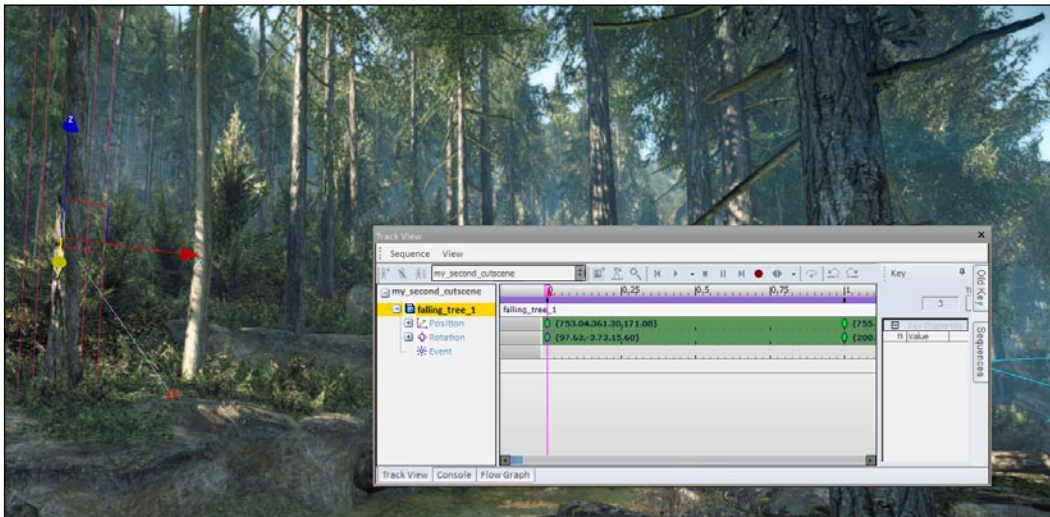
- ▶ You should have `forest.cry` open for this recipe
- ▶ You should have created a new sequence with the name `my_second_cutscene`

How to do it...

First, we must add some geometry to animate. It is important to remember the difference between a brush and the entities. Brushes cannot be added to the Track View.

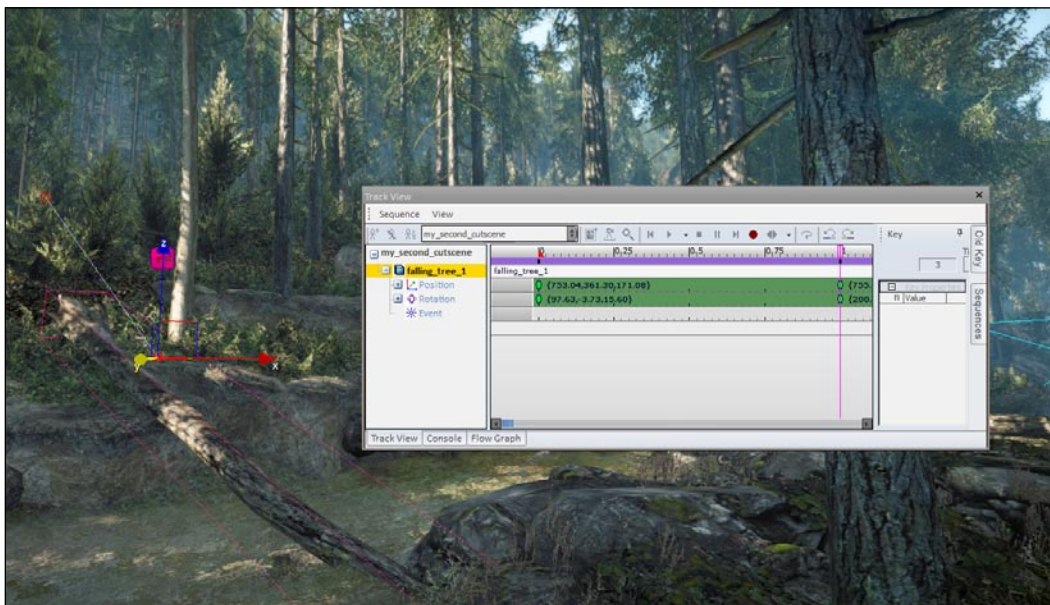
1. To allow us to animate the geometry we are using the basic entity found under the **Entities | Physics** section of the Rollout bar.
2. Once placed, name the **BasicEntity** to **falling_tree_01** and set its model string to `objects/natural/vegetation/rocky_ravine/d_spruce_dead_a.cgf`.

3. Drag the **Animobject** somewhere near the water where it might be able to fall in and allow the player to walk up.



4. Create an initial key frame and then drag the active time slider over to **1** second.
5. Reposition the tree to a fallen pose with the **Record** button set to active.

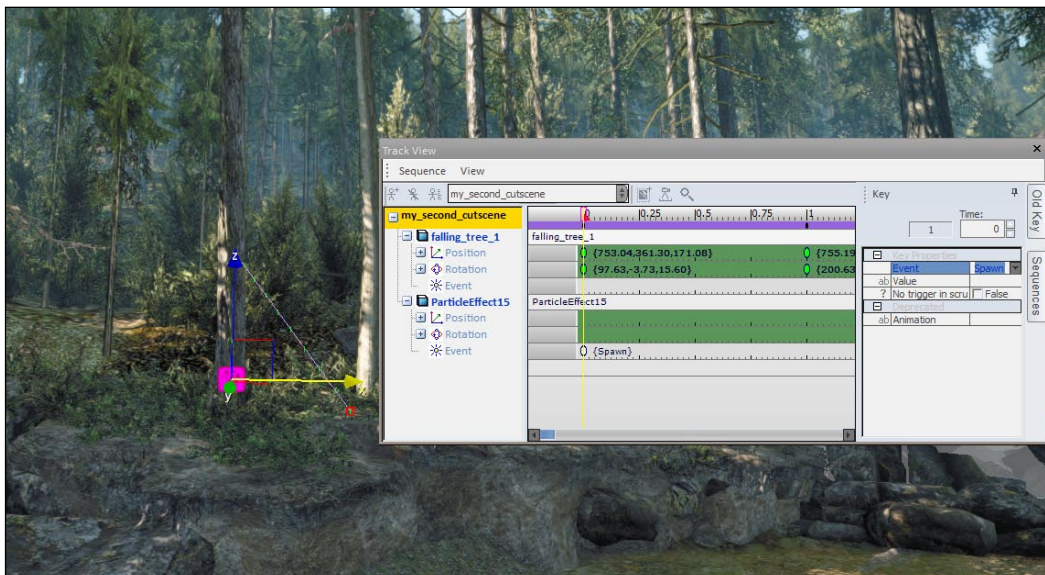
For the ease of this recipe, we will only create two poses but you can of course add more key frames for a better looking animation.





With this kind of animation, you can rely on the preview timing for collision type events such as particle effects. The alternative would be to physically simulate this in the physics engine, which could be good but not deterministic enough to be used for a reliable game play.

- Next, let's add two particle effects to this sequence.
The first will be the breaking of the tree at the base with a wood splintering effect.
- Add a particle effect from the database view under the `particles/breakable_objects/tree_break/small` to the scene and align it to the base of the tree.
- Add this particle effect to the Track View and create a key frame on the event timeline for the particle effect. The property on this key frame must be set to spawn. Create this key frame at the beginning of the scene to coincide with the destruction of the tree.



9. Scrub the timeline until you find the point where the tree impacts the water.
10. Once you have found the first frame in contact between the water and the tree, place a second particle effect from the database view under `particles/water/body_splash/corpse`.

11. Add the spawn key frame for this particle effect just at the contact time between the water and the tree.
12. Play the sequence back to preview the results!

You can now trigger this sequence through a Flow Graph depending on many different events!

How it works...

Animating the entities can allow for designer or animator controlled scripted type events. This allows a fairly advanced manipulation of a wide variety of entities for these events. One important use of this kind of rudimentary animation is to white box certain cinematic events that may be further polished later on.

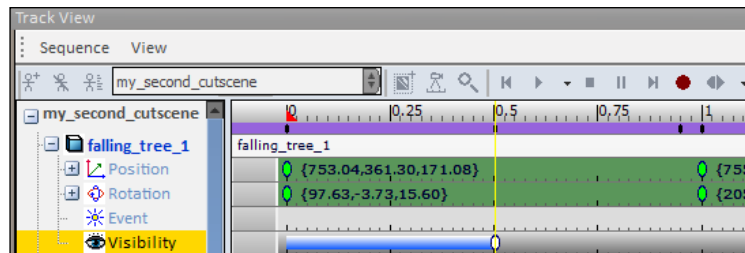
There's more...

You may want to know more about the different tracks available on entities or how to animate the scale of an object.

Entity visibility track

A track available to most entities is the visibility track. We can add this by right-clicking the entity in the sequence and selecting the visibility track.

When a key frame is created on this track, it will change the state of the object to *hidden* or vice versa. The easiest way to visualize it is to see that when the track is blue the object is seen, and when it is not, the object is hidden.



Animating scale

Scale is also accessible to most entities via the Track View. A scale track can be added by right-clicking the added node and selecting scale.

Entities and their tracks

Some entities have unique tracks. For example, characters can play animations and explosion entities can have explode events. Experiment with different entities in your cut-scene to see what is exposed.

See also

- ▶ Go to the *Creating a new Track View sequence* recipe earlier in this chapter to refresh your memory on how to create a new sequence
- ▶ Carry on to the next recipe to learn how to play animations on different entities in the Track View

Playing animations on entities in the Track View

In most cut-scenes, it is far easier to bring in authored animation on characters and even objects from a DCC tool such as 3ds Max.

To be able to play these animations back in the Track View is a valuable tool as it allows for more customization in what they want to achieve.

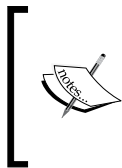
Getting ready

You must have created a new cut-scene in a level and added a camera to it.

How to do it...

Let's play some animation on an entity in the Track View:

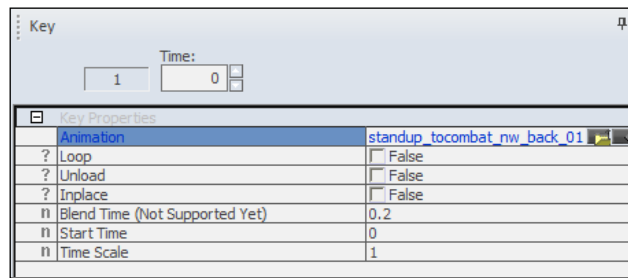
1. Add an object of interest to the scene, for the camera to focus on.
2. In this example, you will add a human character, with a simple animation.



For characters in cinematics, an `AnimObject` is used, as opposed to an AI Entity, for performance reasons. Also, because an `AnimObject` doesn't have AI, the AI system won't conflict/fight with the Track View system.

3. Select the **AnimObject** in the **RollupBar** and drag it into the level. With the **AnimObject** selected, click **Entity Properties** in the **RollupBar** and set the string of the model to a human character.
`objects/characters/neutral_male/sdk_character_male.cdf`
4. In the Track View, click the **Add Selected Node** icon. With the **AnimObject** added, you can now add an animation track to animate the character.
5. Right-click the entity and on the **Add Track** menu, click **Animation**.

6. Select an animation for the character.
7. Double-click at frame **0** to add a key in the event track, and then select the key to bring up the **Key Properties** section.
8. Then, from the drop-down list, select the **standup_tocombat_nw_back_01** animation:



How it works...

The AnimObject entities can have the animation track, which allows us to add pre-authored animations to the entity in the game. This is preferable for many cut-scenes where AI would be inefficient or difficult to perform the exact same movements at the right times.

It is typical to hide an AI entity and then to unhide the AnimObject version of the AI for the cut-scenes.

There's more...

You may want to know how to set up a looping animation or how to adjust the start time and time scale for the animation on the entity.

Loop animation

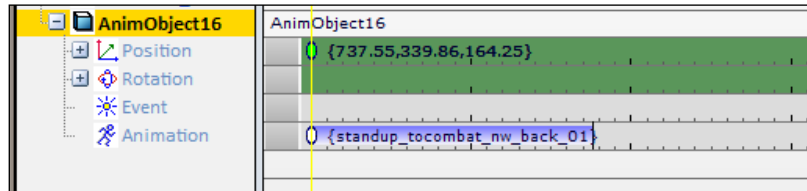
Should you want the animation asset to loop on the character, you can simply set the loop Boolean to `True`. This will loop the asset until told to stop.

Start time

You can manually set a time in the start time dialog to go to a certain frame of the animation. This can be useful for different facial sequences as well as for climbing animations.

Time scale

Using time scale, you can adjust the speed of animation to faster or slower than originally intended. When adjusting the time scale, you will notice that the animation track will extend the length of the selected animation when the loop is not selected.



See also

- ▶ Go to the *Animating entities in the Track View* recipe in this chapter to learn how to animate objects manually
- ▶ You should go to the *Creating a new Track View sequence* recipe earlier in this chapter to learn how to create new sequences

Using console variables (CVars) in the Track View

Now that we have learned to create a variety of different sequences, we will explore some of the bridges that the Track View has designed within to communicate to different systems within the CryENGINE.

Getting ready

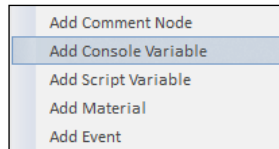
For this recipe, we will add some console commands to the Track View sequence created in the first and the second recipes of this chapter. These two recipes must be already created to go forward.

You should have the cut-scene `my_first_cutscene` open in the Track View and your view set to that of the camera.

How to do it...

Let's learn how to use a console variable key frame in the Track View:

1. Right-click the parent node of your first cut-scene and add a console variable:



2. You will then be asked to name this console variable.

It is very important that you name the console variable with the exact name of the variable you would type into console to achieve the same effect.

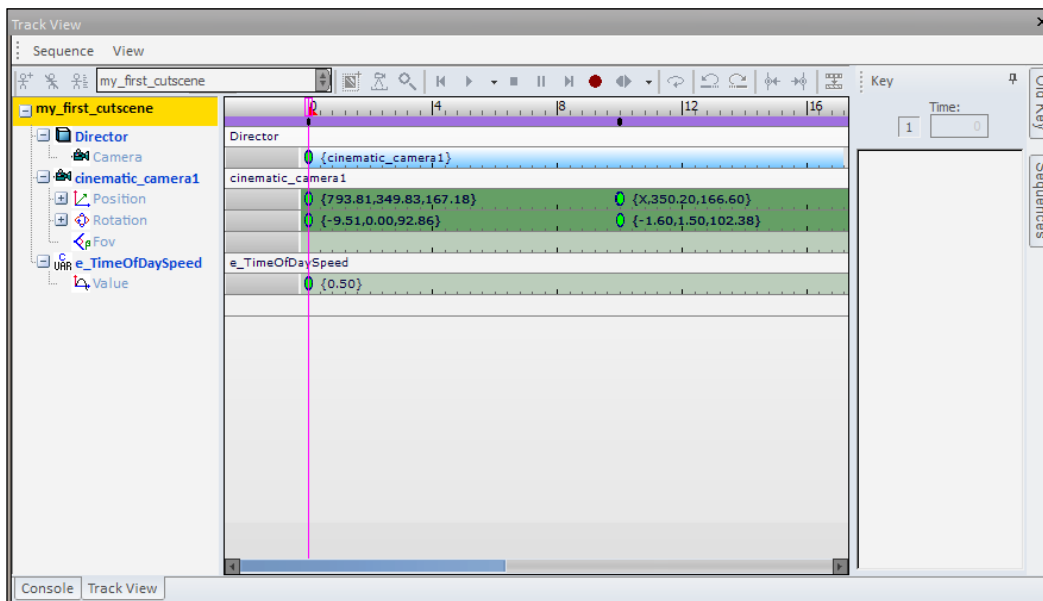
3. For this example, set the name to **e_TimeOfDaySpeed**.

You will see that it adds a track to the parent node of the sequence.

4. Next, create a key frame on the **e_TimeOfDaySpeed** track.

5. Set the value of this key frame to **0.5**.

What this will now do is force a command to the console, in this case, increasing the time of play back speed to 0.5.



6. It should be noted that you cannot preview the CVar events if the sequence is not triggered from within the game.
In this case, you can use the previous setup performed earlier by creating a flow graph and adding a **input:key** node to quickly trigger the sequence.
7. Trigger the flythrough sequence and notice that the *Time of Day* now plays according to the value set in the Track View.

How it works...

Setting the console variables can sometimes be an easy way of achieving some tasks. In this case, you can make a connection between events happening within a track sequence and the engine's console.

There's more...

You may want to know more about animating the CVar values or how to use a powerful CVar for the time scale.

Animating the CVar values

Console variables that are added to the sequence can have many different key frames, which allows for the animation of these values. For example, you may want to change the play speed of the *Time of Day* gradually from fast to slow or vice versa.

T_scale cvar in Track View

Sometimes, to slow down an entire sequence at once the `t_scale` cvar is used. This can be used quite safely as long as it's understood that the default value is 1. A good setting for a slow motion bullet time feel is about 0.2.

See also

- ▶ Go to the *Creating a new Track View sequence* recipe earlier in this chapter to learn how to create a sequence to add console variables to
- ▶ Go to the *Using track events* recipe to learn how to create a special events trigger from the Track View to the Flow Graph

Using track events

This recipe will take you through setting up and using track events in the Track View editor.

A track event is a one-way signal that will allow you to branch Flow Graph logic from a Track View sequence.

Each sequence may define any arbitrary number of track events, which can be called at any time from a *Track Event* node. Each event may also carry with it a string value assigned to the key in the Track View editor. When a track event is triggered from the sequence, its corresponding output port in a special Flow Graph node is activated, allowing you to branch Flow Graph logic very easily.

Getting ready

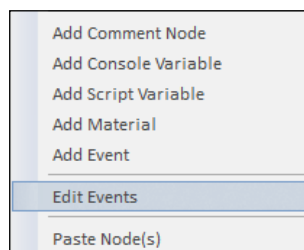
For this recipe, we will add some track events to the Track View sequence created in the first and second recipes of this chapter. These two recipes must be already created to go forward.

You should have the cut-scene `my_first_cutscene` open in the Track View and your view set to that of the camera.

How to do it...

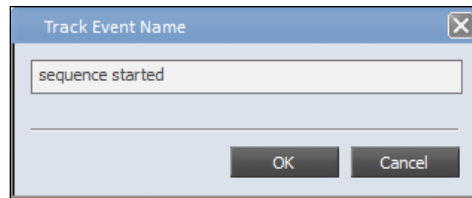
First we must define some track events (track events may be defined per Track View sequence):

1. Open the `my_first_cutscene` sequence, or create a new one to explore track events quickly.
2. Once opened, right-click on the sequence in the tree view on the left and select the **Edit Events** option:



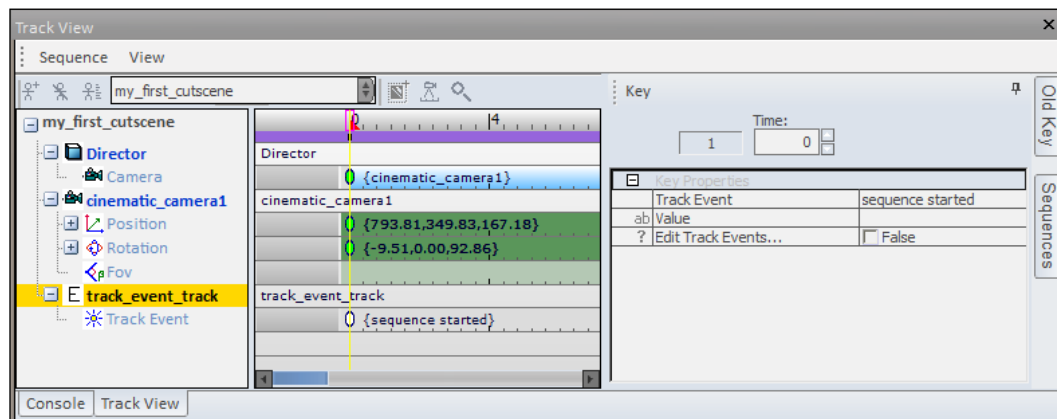
3. A dialog box labeled **Track View Events** will open.
4. To create a track event, select the **Add** button and give the event a name. The event will then be added to the list.

5. To remove this event, select it from the list and select the **Remove** button. When you are done, select the **OK** button to save your changes.
6. Create a new event called `sequence started`:

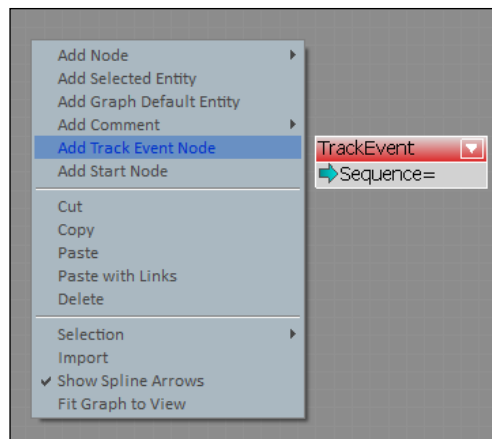


The track event node will allow you to call a track event with an optional string value from the sequence.

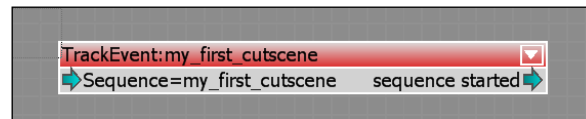
7. To add the node to your sequence, right-click on the sequence in the tree view on the left and select the **Add Event** option towards the bottom.
8. Give the node a name and it will then be added to your sequence.
9. From here, you will be able to add keys to toggle the track events just like any other node.



10. If you select any of the keys you created on the event track, you can assign to it one of your predefined event names by choosing them on the right side menu, called **Key Properties**. You can also edit the list of track events for the sequence quickly by double-clicking on **Edit Track Events**.
11. The next step will be to add this event to a Flow Graph.
12. To add your Flow Graph logic, you will first need to create a new graph or open an existing one.
13. Then you will need to place a track event node, which can be done by right-clicking anywhere in the graph and selecting it from the context menu.



14. The node will initially be empty and you will need to select your animation sequence for the **Sequence** input port.
15. Once you select a valid sequence, the output port will be created for each track event owned by that sequence.



16. From this point, create your Flow Graph logic using the created output ports.

When these events are triggered from the animation sequence as it plays through, the output ports will trigger, allowing your Flow Graph logic to execute.

How it works...

Track Events allow for direct triggering at particular times within a sequence to the Flow Graph. This allows for extremely complex combination of animation and flow graphed physics or effects to be triggered through a sequence!

There's more...

You may want to know how to remove unused track events or how to trigger image nodes for use within the track sequences.

Removing events from the sequence

If after placing a Flow Graph node, you add or remove track events from the sequence, the Flow Graph node will update to reflect this change and the output ports will be created/removed as such.



If you had a link leading from the output port of an event that is later removed, the link will be deleted as well.

Triggering image nodes for track sequences

There are some powerful nodes in the Flow Graph under the image nodes.

These nodes can be triggered from a track event and distort or add some sort of effect to the camera for sequences only.

See also

- ▶ You do not have to create a Flow Graph to trigger console commands, go to the *Using console variables in the Track View* recipe earlier in this chapter

11

Fun Physics

In this chapter, we will cover:

- ▶ Low gravity
- ▶ Hangman on a rope
- ▶ Tornadoes
- ▶ Constraints
- ▶ Wrecking ball
- ▶ Rock slide

Introduction

Up to this point, we have looked into several different methods and tools to achieve some of what's possible with CryENGINE 3. In this chapter, we will look into several different examples, specifically of what the physics engine is capable of, such as manipulating gravity and setting up the different objects with rope physics.

Low gravity

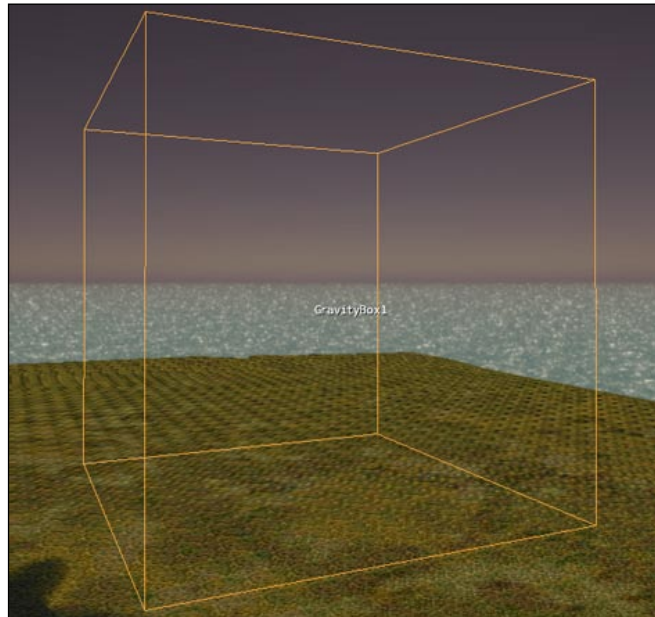
In this simple recipe, we will look at utilizing the *GravityBox* to set up a low gravity area within a level.

Getting ready

- ▶ Have Sandbox open
- ▶ Then open `My_Level.cry`

How to do it...

1. To start, first we must place down a *GravityBox*.
2. In the **RollupBar**, click on the **Entities** button.
3. Under the **Physics** section, select **GravityBox**.
4. Place the **GravityBox** on the ground:



Keeping the default dimensions (20, 20, 20 meters), the only property here that we want to change is the gravity. The default settings in this box set this entire area within the level to be a zero gravity zone. To adjust the up/down gravity of this, we need to change the value of gravity and the **Z** axis.

To mimic normal gravity, this value would need to be set to the acceleration value of -9.81 . To change this value to a lower gravity value, (something like the Moon's gravity) simply change it to a higher negative value such as -1.62 .

How it works...

The *GravityBox* is a simple bounding box which overrides the defined gravity in the code (-9.81) and sets its own gravity value within the bounding box. Anything physicalized and activated to receive physics updates will behave within the confines of these gravitational rules unless they fall outside of the bounding box.

There's more...

Here are some useful tips about the gravity objects.

Uniform property

The `uniform` property within the *GravityBox* defines whether the *GravityBox* should use its own local orientation or the world's. If `true`, the *GravityBox* will use its own local rotation for the gravitational direction. If `false`, it will use the world's direction. This is used when you wish to have the gravity directed sideways. Set this value to `True` and then rotate the *GravityBox* onto its side.

Gravity sphere

Much like the *GravityBox*, the *GravitySphere* holds all the same principles but in a radius instead of a bounding box. The only other difference with the *GravitySphere* is that a false `uniform` Boolean will cause any object within the sphere to be attracted/repulsed from the center of the axis.

Hangman on a rope

In this recipe, we will look at how we can utilize a rope to hang a dead body from it.

Getting ready

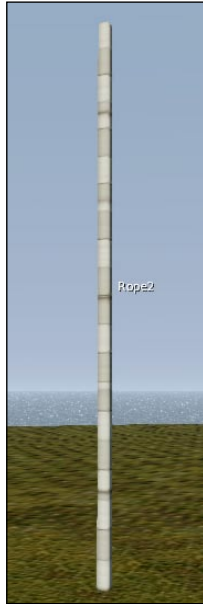
- ▶ Open `Sandbox`
- ▶ Then open `My_Level.cry`

How to do it...

Begin by drawing out a rope:

1. Open the **RollupBar**.
2. From the **Misc** button, select **Rope**.
3. With **Grid Snap** on and set to 1 meter, draw out a straight rope that has increments of one meter (by clicking once for every increment) up to four meters (double-click to finalize the rope).

4. Align the rope so that from end to end it is along the Z axis (up and down) and a few meters off the ground:



5. Next, we will need something solid to hang the rope from.
6. Place down a solid with 1, 1, 1 meter.
7. Align the rope underneath the solid cube while keeping both off the ground. Make sure when aligning the rope to get the end constraint to turn from red to green. This means it is attached to a physical surface:



8. Lastly, we will need to hang a body from this rope. However, we will not hang him in the traditional manner, but rather by one of his feet.
9. In the **RollupBar**, click on the **Entities** button.
10. Under the **Physics** section, select **DeadBody**.
11. Rotate this body up-side-down and align one of his feet to the bottom end of the rope.
12. Select the rope to make sure the bottom constraint turns green to signal that it is attached.
13. Verify that the *Hangman on a rope* recipe works by going into game mode and punching the dead body:



How it works...

The rope is a complicated cylinder that can contain as many bending segments as defined and is allowed to stretch and compress depending on the values defined. Tension and breaking strength can also be defined. But since ropes have expensive physics properties involved, they should be used sparingly.

See also

- ▶ The *Wrecking ball* recipe

Tornadoes

Another fun physics entity is the *Tornado*. In this recipe, we will look at placing one down and allowing this tornado to roam around the level. We will also demonstrate it picking up and throwing around multiple AI Grunts.

Getting ready

- ▶ Open `My_Level.cry` within the Sandbox
- ▶ Place down a dozen AI Grunts

How to do it...

Place down the *Tornado* entity by doing the following:

1. In the **RollupBar**, click on the **Entities** button.
2. Under the **Environment** section, select **Tornado**.
3. The default values for the tornado will not be sufficient to lift up the AI Grunt, so we will have to adjust the following values:

SpinImpulse = 40

UpImpulse = 40

AttractionImpulse = 100

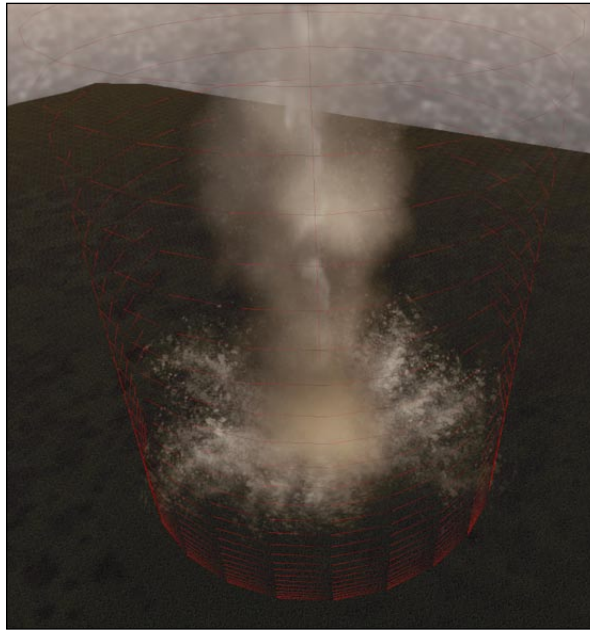
With these properties set, the next time we enter into game mode, the tornado will start to wander around and pick up and toss the AI Grunts. Even the player himself can get caught in this and be pulled around by the tornado.

How it works...

The *Tornado* entity is a special entity that applies multiple forces into a defined radius around the entity. These forces are the same forces as described previously with **SpinImpulse**, **UpImpulse**, and **AttractionImpulse**. You can also add in a *WanderSpeed*, which will allow the tornado to wander around in a random direction that will still follow the terrain.

There's more...

The `p_draw_helpers` property, when set to 1, debugs the funnel of the tornado. As the particle of the tornado does not scale with the physics of the tornado itself, you may see a debug rendering of the radius and the spline of the tornado by using `p_draw_helpers = 1`.



Constraints

Previously in the *Hangman on a rope* recipe, we've looked at ropes and how they can bend and move with the object(s) they are bound to. In this recipe, we will look at constraints, which are a much cheaper *hinge* that can be used on two objects. Specifically, we will hang a primitive pyramid object from a solid and allow it to swing on all axes.

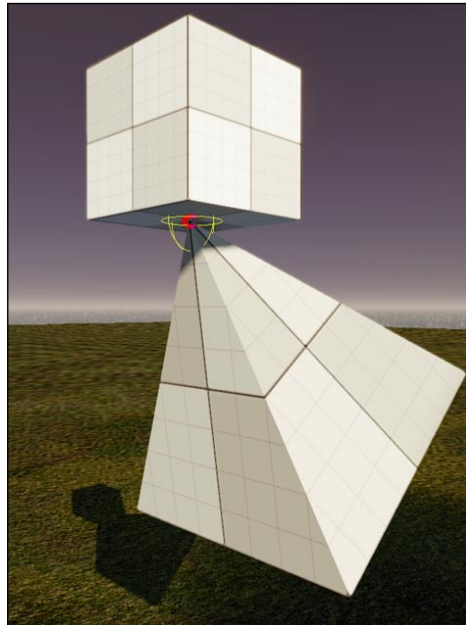
Getting ready

- ▶ Open `My_Level.cry` within the Sandbox
- ▶ Place down a 1x1x1 meter solid and suspend it in the air roughly four meters off the ground
- ▶ Place down a **BasicEntity** and change the model to `objects/default/primitive_pyramid.cgf`
- ▶ Set the mass of the **BasicEntity** to 100

How to do it...

Begin by placing down a constraint:

1. In the **RollupBar**, click on the **Entities** button.
2. Under the **Physics** section, select **Constraint**.
3. Place the constraint on the bottom surface of the solid box.
4. Move the pyramid underneath the solid box and have the tip intersect with the radius of the constraint.
5. The next time the player jumps into the game and pushes the pyramid, it will move according to the constraint provided.



How it works...

A constraint entity can create a physical constraint between two objects. The objects will be selected automatically during the first update, by sampling the environment in a sphere around the constraint object's world position with a specified radius. The *first* object (the one that will own the constraint information internally) is the lightest among the found objects, and the second is the second lightest (static objects are assumed to have infinite mass, so a static object is always heavier than a rigid body).

Constraints operate in a special **constraint frame**. It can be set to be either the frame of the first constraint object (if **UseEntityFrame** is checked), or the frame of the constraint entity itself. In that frame, the constraint can operate either as a hinge around the **x** axis, or as a ball-in-a-socket around the **y** and **z** axes (that is, with the **x** axis as the socket's normal). If **x** limits are set to a valid range ($\text{max} > \text{min}$) and the **yz** limits are identical (such as both ends are 0), it is the former and if the **yz** limits are set and not **x** limits, it's the latter. If all limits are identical (remains 0, for instance), the constraint operates in three degrees of freedom mode (that is, it doesn't constrain any rotational axes). If all limits are set, no axes are locked initially, but there are rotational limits for them.

There's more...

Here are the definitions of the properties on the constraints.

Constraint properties

The following are the constraint properties:

- ▶ **Damping:** This sets the strength of the damping on an object's movement. Most objects can work with 0 damping; if an object has trouble coming to rest, try values such as 0.2-0.3. Values of 0.5 and higher appear visually as overdamping. Note that when several objects are in contact, the highest damping is used for the entire group.
- ▶ **max_bend_torque:** This is the maximum bending torque (currently it's only checked against for hinge constraints that have reached one of the X limits).
- ▶ **max_pull_force:** This specifies the maximum stretching force the constraint can withstand.
- ▶ **NoSelfCollisions:** This disables collision checks between the constrained objects (to be used if the constraint is enough to prevent inter-penetrations).
- ▶ **Radius:** This defines spherical area to search for attachable objects.
- ▶ **UseEntityFrame:** This defines whether to use the first found object or the constraint itself as a constraint frame.

Wrecking ball

Instead of using the dead body that we used in the *Hangman on a rope* recipe, we will look at attaching a heavy *RigidBodyEx* in the shape of a ball to make a wrecking ball. Through this the player will be able to activate the rope physics to swing the wrecking ball into a breakable house and destroy a section of wall on it.

Getting ready

- ▶ You should have completed the *Hangman on a rope* recipe
- ▶ Open `My_Level.cry` within the Sandbox

How to do it...

Using what we've already learned from the *Hangman on a rope* recipe, we will now swap out the *DeadBody* for a *RigidbodyEx*:

1. In the **RollupBar**, click on the **Entities** button.
2. Under the **Physics** section, select **RigidbodyEx**.
3. After placing the **RigidbodyEx** into your level, you first want to change the model over to a sphere.
4. Select the newly placed **RigidbodyEx**.
5. In the **RollupBar**, you will find **Model** under **Entity Properties**.
6. Change the model to the following `.cgf` model `objects/default/primitive_sphere.cgf`.
7. Then change the following properties still found under **Entity Properties**:

CanBreakOthers = true

Mass = 1000

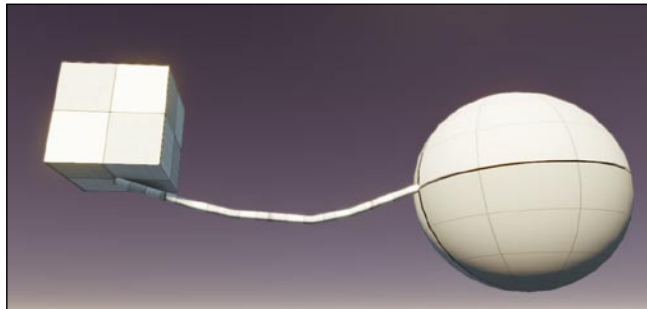
Resting = False

8. Next, we will need to adjust our vertical rope to line up more horizontally with a bit of a bend in it.

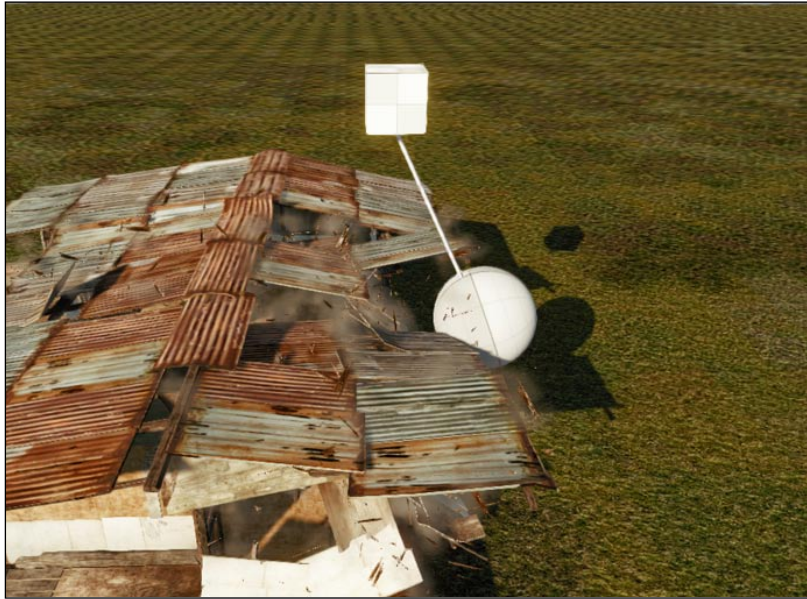


Select your rope and use **Edit Rope** to move the rope segments along the Y and Z axes.

9. Lastly, line up the *RigidbodyEx* to the end of the rope. The final assembly of this should look like the following screenshot:



Now the wrecking ball is ready. However, it needs something to destroy. Place down a destructive object in the path of the wrecking ball such as the following brush `Objects/library/architecture/village/village_house1_c.cgf`. After **setting up the** wrecking ball over the head of the house, the player will see destruction occur the next time they enter the level.



How it works...

The setup of the wrecking ball is very similar to the hangman, but just used in a different context of how designers can utilize the rope physics.

See also

- The *Hangman on a rope* recipe

Rock slide

In the wrecking ball recipe, we looked at using a *RigidBodyEx* sphere model to use as our wrecking ball. In this recipe, we will use that same *RigidBodyEx* as the basis for a rock analog and clone it a few times so we can build our landslide.

Getting ready

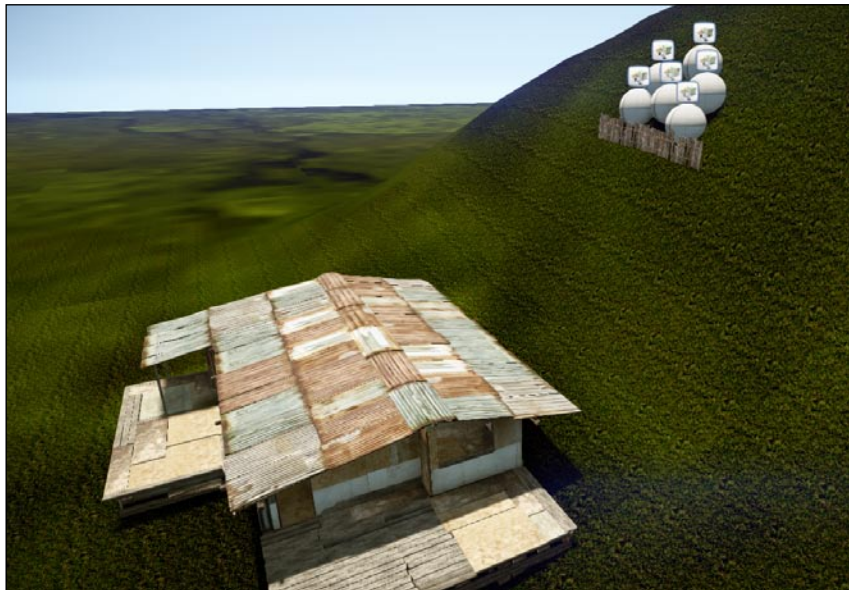
- ▶ You should have completed the *Wrecking ball* recipe
- ▶ Open `My_Level.cry` within the Sandbox

How to do it...

1. Begin by creating a terrain hill about 30 meters in height with a slope of about 35-45 degrees.
2. Copy and paste the house that was used from the *Wrecking ball* recipe and place it at the bottom of the hill.
3. Next, select the sphere *RigidBodyEx* from the *Wrecking ball* recipe and clone it and place it more than half way up the hill. Also, change the following properties on the sphere:

Resting = False

4. Next, clone the *RigidBodyEx* five times and stack them on the hill like a carton of eggs. Make sure they are tight and are also touching each other (this will make sure they all get activated physics when triggered).
5. Last, place the breakable brush fence object `Objects/library/barriers/fence/wooden/wooden_fence_120_400_a.cgf` under the bottom *RigidBodyEx*. Be sure that the fence is also touching the *RigidBodyEx*s.
6. Your setup should look like the following screenshot:



7. The next time the player enters the level and destroys the fence placed with something like a rifle, the *RigidBodyEx* will activate its physics and slide/roll down the hill and destroy the house as shown in the following image:



How it works...

The *RigidBodyEx* entity is an extension to the *BasicEntity* that reacts more realistically with other entities and water. The attached object can break other objects and has more options as to when and how to react.

There's more...

Here are some additional properties that can be applied to the *RigidBodyEx* class of the entity.

Physics properties

The physics properties are as follows:

- ▶ **ActivateOnDamage:** This tells us that an inactive rigid body (*RigidBodyActive*=0) should be activated on damage.
- ▶ **CanBreakOthers:** This is *True* if the entity can break joined objects by colliding with them (provided they overcome the strength limit). Basic entities have this flag off unconditionally.
- ▶ **Density:** ($\text{Density} = \text{Volume} / \text{Mass}$) This affects the way objects interact with other objects and float in the water (they sink if their density is more than that of the water). Note that both mass and density can be overridden in the asset.
- ▶ **Mass:** ($\text{Mass} = \text{Density} * \text{Volume}$) This is the weight of the object (the density of the object multiplied by its volume).

- ▶ **Physicalize:** If not set, the object will not be taken into account by physics.
- ▶ **PushableByPlayers:** When set, the object can be moved by the player.
- ▶ **Resting:** When `True`, the object's physics are asleep on start. When `False`, the physics are activated until kinetic energy falls below the `sleep_speed` limit and is put to sleep again (refer to *sleep_speed* shown further).
- ▶ **RigidBody:** `False` means a static entity; `True` means a simulated rigid body. Note that a rigid body can still behave like a static entity if it has mass 0 (set either explicitly or by unchecking *RigidBodyActive*). The main difference between these rigid bodies and pure statics is that the physics system knows that they can be moved by some other means (such as the *Track View*) and expects them to do so. This means that objects that are supposed to be externally animated should be mass-0 rigid bodies in order to interact properly with pure physicalized entities.
- ▶ **RigidBodyActive:** This property indicates that the object is a rigid body, but initially it is immovable. Instead, it can be activated by an event later.

Simulation properties

The simulation properties are as follows:

- ▶ **Damping:** (0..3) This sets the strength of the damping on an object's movement. Most objects can work with 0 damping; if an object has trouble coming to rest, try values like 0.2-0.3. Values of 0.5 and higher appear visually as overdamping. Note that when several objects are in contact, the highest damping is used for the entire group.
- ▶ **FixedDamping:** (true/false) When true, this object will force its damping to the entire colliding group (use it when you don't want a particular object being slowed by a highly damped entity, such as a dead body).
- ▶ **max_time_step:** (0.005..0.1) This sets the maximum time step the entity is allowed to make (defaults to 0.01). Smaller time steps increase stability (can be required for long and thin objects, for instance), but are more expensive. Each time the physical world is requested to make a step, the objects that have their *maxsteps* smaller than the requested one, slice the big step into smaller chunks and perform several sub steps. If several objects are in contact, the smallest *max_time_step* is used.
- ▶ **sleep_speed:** (0.01..0.3) If the object's kinetic energy falls below some limit over several frames, the object is considered sleeping. This limit is proportional to the square of the sleep speed value. A sleep speed of 0.01 loosely corresponds to the object's center moving at a velocity of the order of 1 cm/s.

See also

- ▶ The *Wrecking ball* recipe

12

Profiling and Improving Performance

In this chapter, we will cover:

- ▶ Profiling performance in the Sandbox
- ▶ Saving level statistics
- ▶ Enabling debug draw modes
- ▶ Optimizing the levels with VisAreas and portals
- ▶ Using light boxes and light areas
- ▶ Activating and deactivating the layers

Introduction

In this very important chapter, we will explore some of the techniques used when optimizing the performance of your game or simulation in CryENGINE 3.

In this chapter, you will learn how to measure the performance using some of the tools available to the developer out of the box with the CryENGINE. We will also explore certain printouts that can be made, which will help you profile different performance losses in your level. Finally, we will explore real examples of optimizations that you can make to immediately improve the overall performance of the CryENGINE.

Profiling performance in the Sandbox

CryENGINE has a huge variety of different built-in debugging and profiling tools. As you will soon find out, some of them are very specific to a certain subsystem and only useful for advanced users of the technology.

Others tools, however, are very useful for most engine users in their regular daily workflow. This recipe lists the profiling console variables and commands that each programmer, artist, and level designer working with CryENGINE should know.

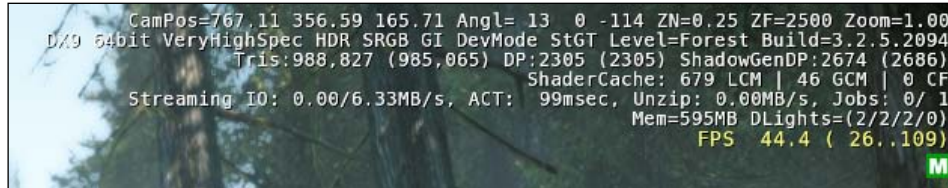
Getting ready

To demonstrate the uses of the profiling tools it is not imperative that you have any particular level open; however, to see real data you should have opened a populated level.

How to do it...

The proverbial first line of defense for developers in profiling and reading performance in the Sandbox is the *display info console* command. It is on by default in the Sandbox.

Notice that you can enable and disable displaying the debug information through typing `r_displayinfo 0` or `1` into the console.



This command displays important information concerning your overall performance. Some of the major values that you should be able to identify are:

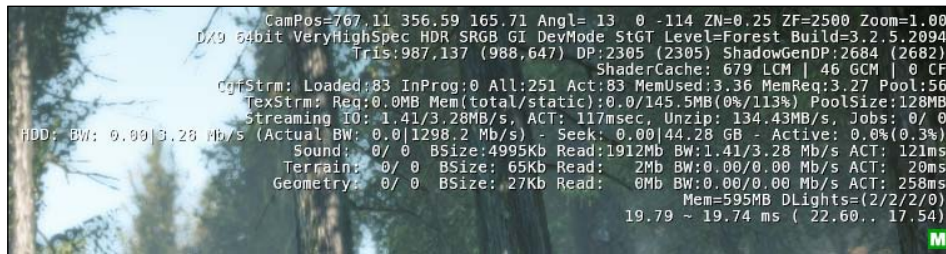
- ▶ **CamPos:** Camera position in world coordinates X, Y, and Z
- ▶ **Tris:** Currently rendered triangle count
- ▶ **DP:** Draw Primitive or Draw Calls
- ▶ **Mem:** Currently allocated memory
- ▶ **DLights:** Dynamic and deferred lights
- ▶ **FPS:** Frames Per Second

These values are important because typically the entire project sets budgets around these values. Other than the camera position if you notice abnormal values or activities in any of these values, you can be sure that you are experiencing some sort of performance loss. The most effective measurement of performance is **frame time**. Frame time refers to the time that each frame takes from the beginning to the end and is usually expressed in milliseconds (ms).

The reason for this is that **Frames Per Second** or **FPS** is defined as $1/\text{frame time}$ and is hence a non-linear measure. To put this into perspective, this means an increase of two FPS when the game running at 20 FPS, gives an extremely profitable gain of five ms, while the same two FPS improvement on a game running at 60 FPS, will just result in a gain of only 0.5 ms.

By setting `r_DisplayInfo` to 2 instead of 1, it is possible to see the frame time instead of the FPS. Here are some useful conversions:

- ▶ 16 ms = 60 FPS
- ▶ 33 ms = 30 FPS (target for CryENGINE)
- ▶ 40 ms = 25 FPS



How it works...

Performance can heavily depend on the run-time environment. It is thus quite important to use similar conditions when comparing performance numbers on different levels and on different system specifications. The performance on systems with different hardware, for example, is likely to vary a lot. The GPU time is also very dependent on screen resolution. The higher the resolution used, the lower the overall performance will become. Being able to accurately measure and read the performance of your game in Sandbox and from the launcher allows you to set budgets and follow them when creating the content.



The **Display Info** tab contains enough high-level information for general performance profiling.

There's more...

You will surely want to know what constitutes a Draw Call and where their values and costs come from, as well as how to deal with budgets and triangle counts.

Draw Calls

Every object with a material has a separate Draw Call. This also means that every sub material in a material is a separate Draw Call.

Each Draw Call means setting material data and some other extra work on the CPU side plus fill rate costs on the GPU side. This varies depending on screen area occupied by the Draw Call itself.

Draw Call count will be affected depending on a certain number of conditions:

- ▶ Opaque geometry has at least two Draw Calls: one for the *Zpass* otherwise known as the **Depth Pass**, and the other for general rendering.
- ▶ Shadows are extra Draw Calls, as are detail passes. There may even be material layers such as wet or frozen that will add to this count.
- ▶ The amount of non-deferred lights affecting geometry.

Strive to maintain an acceptable Draw Call count, which is typically around 2000 Draw Calls. This can be done easily on design as well as art side and should be monitored at all times on a per view basis.

Triangle count

One way to view just the triangles in CryENGINE is to enable wireframe. You can do this using the console variable `r_wireframe = 1`.

This visualizes what the display information is saying it is currently drawing for triangles. This should be monitored and can usually identify faults in occlusion as well as view distance ratios.

Budgets

In most cases, it is impossible to define specific per-object budgets for each art asset - the final makeup of each view in the game depends on too many variables (how many objects, how many characters, how much vegetation, and so on).

For example: A good overall budget should have:

- ▶ 2000 Draw Calls
- ▶ 1 million triangles

These budgets represent final in-game maximums for any particular view, including all rendering features and effects (after the entire scene has been polished).

These budgets do not give the environment or character artists a clear budget for how many triangles to put into a building or a character but rather define how much can be viewed at one time. It can be used by level artists to determine how many whitebox buildings should go into a particular view and still leave enough room for characters, vehicles, and even particles.

See also

- ▶ Go to the *Creating a new level* recipe in *Chapter 2, Sandbox Basics* to create your own level to be profiled
- ▶ Carry onto the next recipe to find out more about profiling level performance with a printout

Saving level statics

In this recipe, we will create an XML report with statistics for the currently loaded level. The report includes all assets that are loaded, their size, dependencies, and the number of instances in the scene.

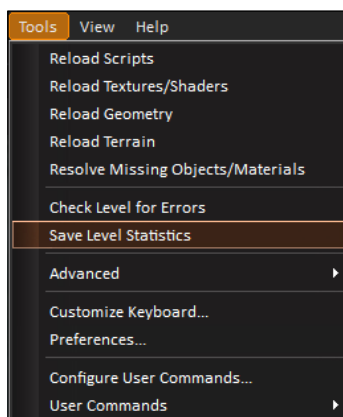
Getting ready

The report that we will generate will be created as an `.xml` and can be opened by using Microsoft Excel.

How to do it...

The **Save Level Statistics** printout is accessible in two main ways. Both are quite simple:

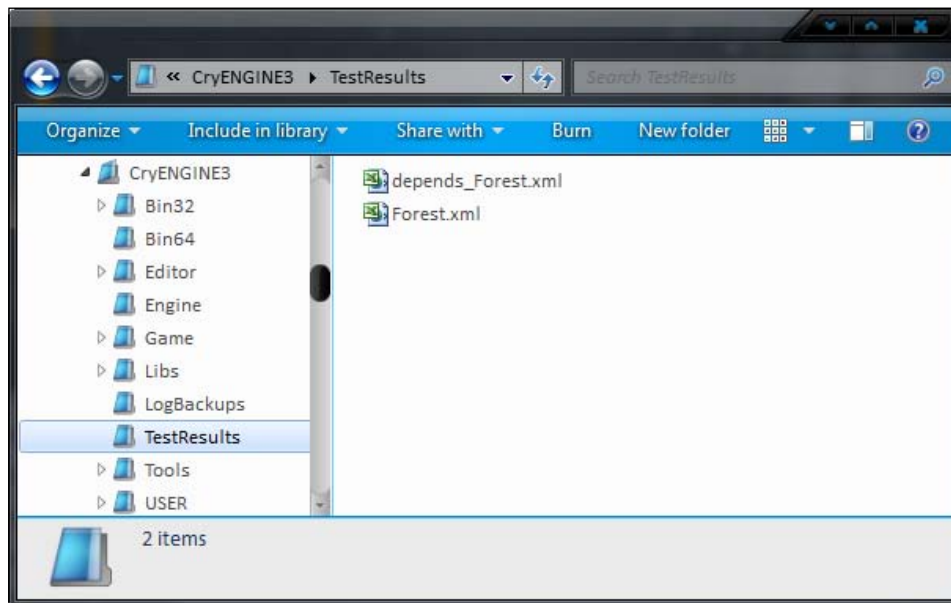
1. The first way is to use the **Tools** menu in the Sandbox and navigate to the **Save Level Statistics** option:



2. The second way is to output the level statistics by using a console command. Triggering the writing of the data from the console is quite useful when running in the launcher. The console command you must enter is `SaveLevelStats`.

Performing this command through either method will create two `.xml` files under the `rootdirectory/testresults` folder.

The names of these files contain the names of the level that they were saved from. For example, the two files in the following example were saved from `forest.cry`:



These files are best opened with Microsoft Excel. Open `Forest.xml` in Excel. The first page of the **Save Level Statistics** printout gives some important information:

The screenshot shows an Excel spreadsheet titled 'Forest.xml - Microsoft Excel'. The spreadsheet contains performance data for CryEngine Version 3.2.5.2094. The data is organized into sections: General, Resource Type (MB), Textures, Meshes, and Memory Usage. The 'Textures' section is highlighted in green. The 'Meshes' section is also highlighted in green. The 'Memory Usage' section is highlighted in green. The 'Textures' section includes a table with columns: Resource Type (MB), Count, Memory Size, Only Mesh Size, and Only Texture Size. The 'Meshes' section includes a table with columns: Resource Type (MB), Count, Memory Size, Only Mesh Size, and Only Texture Size. The 'Memory Usage' section includes a table with columns: Resource Type (MB), Count, Memory Size, Only Mesh Size, and Only Texture Size.

CryEngine Version 3.2.5.2094					
	A	B	C	D	E
1	CryEngine Version 3.2.5.2094				
2	Level Forest				
3	Running in 64bit version				
4	Level Load Time (sec):	0			
5					
6	Average\Min\Max (fps):	54	47	65	
7					
8	Resource Type (MB)	Count	Memory Size	Only Mesh Size	Only Texture Size
9	CGF Objects	175	61	4	56
10	Character Models	16	49	2	46
11	Character Instances	53			
12	Entities	156			
13					
14	Textures				
15	Count	704	Total count of textures		
16	Textures Overall Size	343	Total amount of textures memory usage		
17	Pool Size	130	Size of textures pool		
18	Textures Memory Usage	473	Total memory of textures in RAM		
19	Textures Engine Only	197	Textures for internal Engine usage		
20	User Textures	145	User textures not stored in the textures pool		
21	Textures streaming throughput(KB/s)				
22					
23	Meshes				
24	Count	475			
25	Total Size	13			
26	System Memory	5			
27	Video Memory	8			
28					
29					
30					
31	Lua Memory Usage (MB)	8			
32	Game Memory Usage (MB)	268			
33	Total Allocated (With Code/Textures/Mesh)	659			
34					
35	Virtual Memory Usage (MB)	596			
36	Peak Virtual Memory Usage (MB)	604			
37					
38					
39					
40					

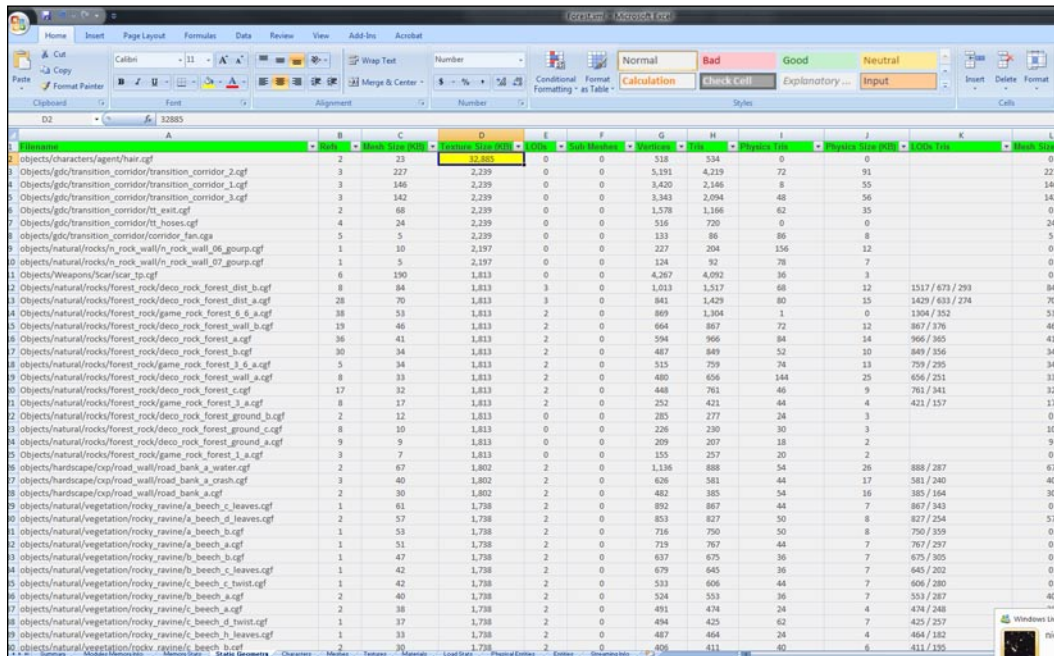
It describes the engine version, the level, and whether the engine is running in a 32 or 64 bit configuration. It also shows the average minimum and maximum frames per second.

The overall cost of a level can be quickly observed here as the count for static CGF geometry is printed as well as for character models and finally entities. It even shows the memory consumption required for each type.

Profiling and Improving Performance

You can also get various profile information through the different pages generated. You can change these pages in Excel easily by clicking the corresponding tab. Click on the **Static Geometry** tab. In this tab, you can see in depth information on every static piece of geometry in the level.

You can arrange the worksheet by using Excel's arrange feature to make it, so it sorts objects from highest to lowest in a certain column. In the following example, you can easily see a performance issue where a hair object is taking up a much greater amount of texture memory than it should:



Object	Mem	Tex Mem	Phys Mem	Phys Tex	Phys Mem	Phys Tex	Phys Mem	Phys Tex	Phys Mem	Phys Tex
Objects/characters/agents/hair.cgf	2	23	1,813	0	0	518	518	0	0	0
Objects/gdc/transition_corridor/transition_corridor_3.cgf	3	227	2,239	0	0	5,191	4,219	72	91	227
Objects/gdc/transition_corridor/transition_corridor_1.cgf	3	146	2,239	0	0	3,420	2,146	8	55	146
Objects/gdc/transition_corridor/transition_corridor_3.cgf	3	142	2,239	0	0	3,343	2,094	48	56	142
Objects/gdc/transition_corridor/n_exit.cgf	2	68	2,239	0	0	1,578	1,166	62	35	68
Objects/gdc/transition_corridor/ht_hoses.cgf	4	24	2,239	0	0	516	720	0	0	24
Objects/gdc/transition_corridor/corridor_fan.cga	5	5	2,239	0	0	113	86	86	8	5
Objects/natural/rocks/n_rock_wall/n_rock_wall_06_group.cgf	1	10	2,197	0	0	227	204	156	12	0
Objects/natural/rocks/n_rock_wall/n_rock_wall_07_group.cgf	1	5	2,197	0	0	124	92	78	7	0
Objects/Weapons/scar/scar_tp.cgf	6	190	1,813	0	0	4,267	4,092	36	3	0
Objects/natural/rocks/forest/rock/deco_rock_forest_dist_b.cgf	8	84	1,813	3	0	1,013	1,517	68	12	1517 / 679 / 293
Objects/natural/rocks/forest/rock/deco_rock_forest_dist_a.cgf	28	70	1,813	3	0	841	1,429	80	15	1429 / 633 / 274
Objects/natural/rocks/forest/rock/game_rock_forest_6_a.cgf	38	53	1,813	2	0	869	1,304	1	0	1304 / 352
Objects/natural/rocks/forest/rock/deco_rock_forest_wall_b.cgf	19	46	1,813	2	0	664	867	72	12	867 / 376
Objects/natural/rocks/forest/rock/deco_rock_forest_a.cgf	36	41	1,813	2	0	594	966	84	14	966 / 365
Objects/natural/rocks/forest/rock/deco_rock_forest_b.cgf	30	34	1,813	2	0	487	849	52	10	849 / 356
Objects/natural/rocks/forest/rock/game_rock_forest_3_a.cgf	5	34	1,813	2	0	515	759	74	11	759 / 295
Objects/natural/rocks/forest/rock/deco_rock_forest_wall_a.cgf	8	33	1,813	2	0	480	656	144	25	656 / 251
Objects/natural/rocks/forest/rock/deco_rock_forest_c.cgf	17	32	1,813	2	0	448	761	46	9	761 / 341
Objects/natural/rocks/forest/rock/game_rock_forest_3_a.cgf	8	17	1,813	2	0	252	421	44	4	421 / 157
Objects/natural/rocks/forest/rock/deco_rock_forest_ground_b.cgf	2	12	1,813	0	0	285	277	24	3	0
Objects/natural/rocks/forest/rock/deco_rock_forest_ground_c.cgf	8	10	1,813	0	0	226	230	30	3	0
Objects/natural/rocks/forest/rock/deco_rock_forest_ground_a.cgf	9	9	1,813	0	0	209	207	18	2	0
Objects/natural/rocks/forest/rock/game_rock_forest_3_a.cgf	3	7	1,813	0	0	155	257	20	2	0
Objects/hardscape/csp/road/wall/road_bank_a_water.cgf	2	67	1,802	2	0	1,136	888	54	26	888 / 287
Objects/hardscape/csp/road/wall/road_bank_a_crash.cgf	3	40	1,802	2	0	626	581	44	17	581 / 240
Objects/hardscape/csp/road/wall/road_bank_a_cg	2	30	1,802	2	0	482	385	54	16	385 / 164
Objects/natural/vegetation/rocky_ravine/a_beech_c_leaves.cgf	1	61	1,798	2	0	892	867	44	7	867 / 343
Objects/natural/vegetation/rocky_ravine/a_beech_d_leaves.cgf	2	57	1,798	2	0	853	827	50	8	827 / 254
Objects/natural/vegetation/rocky_ravine/a_beech_b_cg	1	53	1,798	2	0	716	750	50	8	750 / 359
Objects/natural/vegetation/rocky_ravine/a_beech_a_cg	1	31	1,798	2	0	719	767	44	7	767 / 297
Objects/natural/vegetation/rocky_ravine/b_beech_b_cg	1	47	1,798	2	0	637	675	36	7	675 / 305
Objects/natural/vegetation/rocky_ravine/b_beech_c_leaves.cgf	1	42	1,798	2	0	679	645	36	7	645 / 202
Objects/natural/vegetation/rocky_ravine/c_beech_c_twist.cgf	1	42	1,798	2	0	533	606	44	7	606 / 280
Objects/natural/vegetation/rocky_ravine/b_beech_a_cg	2	40	1,798	2	0	534	553	36	7	553 / 287
Objects/natural/vegetation/rocky_ravine/c_beech_a_cg	2	38	1,798	2	0	491	474	24	4	474 / 248
Objects/natural/vegetation/rocky_ravine/c_beech_d_twist.cgf	1	37	1,798	2	0	494	425	62	7	425 / 257
Objects/natural/vegetation/rocky_ravine/c_beech_b_leaves.cgf	1	33	1,798	2	0	487	464	24	4	464 / 182
Objects/natural/vegetation/rocky_ravine/c_beech_b_cg	2	30	1,798	2	0	406	411	82	6	411 / 155

There are many more tabs and values associated with each. Some tabs will only be useful to people working with code, such as, module memory. Others will be important for artists, such as static objects and characters. Close the `Forest.xml` and open the `depends_Forest.xml`.

The entire file prints out the dependencies of certain files on other files. For example, it shows that a certain `.mtl` might have six different textures depending on it.

In a lot of cases, this can be used to reduce build and distribution sizes as well as identify areas and objects where textures might be able to be combined.

How it works...

The **Save Level Statistics** prints out a variety of information in regards to a level as well as the asset within certain levels. Being able to now print out this information into an Excel spreadsheet allows complex operations to take place, such as tracking performance of a certain level over its development or even identifying problem assets.

As it is in an Excel sheet, further systems could be created within Excel to graph the improvement of level performance over time as it is developed.

There's more...

There are many different tabs presented to you in the **Save Level Statistics** printout. You will want to know more about some of the values within it.

Textures tab and render targets

In the **Textures** tab of the **Save Level Statistics** printout, a common practice is to sort the entire list from highest to smallest in order of texture memory.

When doing this you will inevitably notice values such as `$HDRTarget` or `$SceneTarget`.

Any textures with a `$` proceeding them are known as **render targets**. These are not authored textures but rather scene render targets that are generated by using different shaders and post effects in real-time graphics. The texture size of the render target will depend in most cases on the current render window size.

Physics tris and physics size

In most geometry related printouts, you will see values for both the number of triangles not only a particular objects physics mesh is contained but also the physical size in kilobytes (KB). This can be a very easy place to find problems with physics meshes and performance, as almost all objects need a physics mesh and it must be as small as possible. The other modifier of physics size is the user-defined properties contained within the asset.

Detailed dependencies tab

In the `depends_Forest.xml`, you should have noticed the **Detailed Dependencies** tab. This tab is arguably the most useful as it breaks down thoroughly which assets depend on what. It also simplifies further analysis by adding a prefix for every object with or without dependencies.

This view breaks down the per-frame statistics of the renderer. This is extremely important as the render takes up the highest percentage in most cases of the frame time. It is also the most important view to use when following certain budget guidelines as designers and artist have direct control over these variables by adjusting the number of lights in their viewable scenes or even decals.

2. The next view will allow you to asses the Draw Calls created by different effects as well as by materials on objects. With the advanced deferred lighting solution within CryENGINE, you can have a large number of deferred lights to achieve advanced effects with minimal effect on the Draw Call count. However, it is still very valuable to be able to view the breakdown of Draw Calls by object within a scene.
3. Enable this by using `r_stats 6`.



4. This will show the Draw Call cost of each object on the screen. They are listed by *depthpass* calls, *general* calls, *shadows* calls, and finally *misc*.

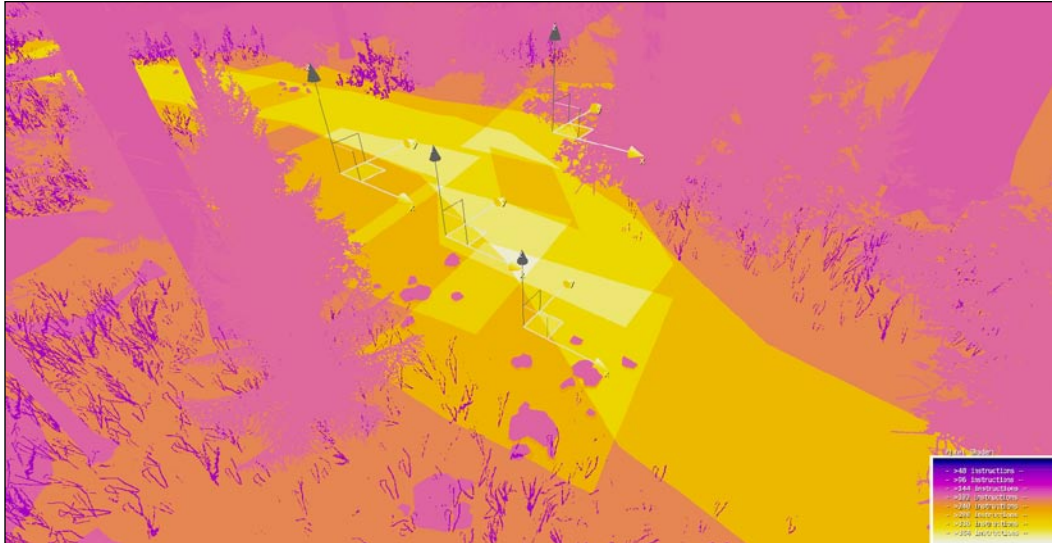


Submaterials add to the Draw Call count if they are used as a render material.

Another important consideration here that could be noted would be to use normal maps for decals as they add to the depth pass calls.

Finally, we will explore `overdraw`:

1. Disable `r_Stats 6` by typing `r_stats 0` into the console.
2. Enable the `overdraw` render mode by typing `r_measureoverdraw 1`:

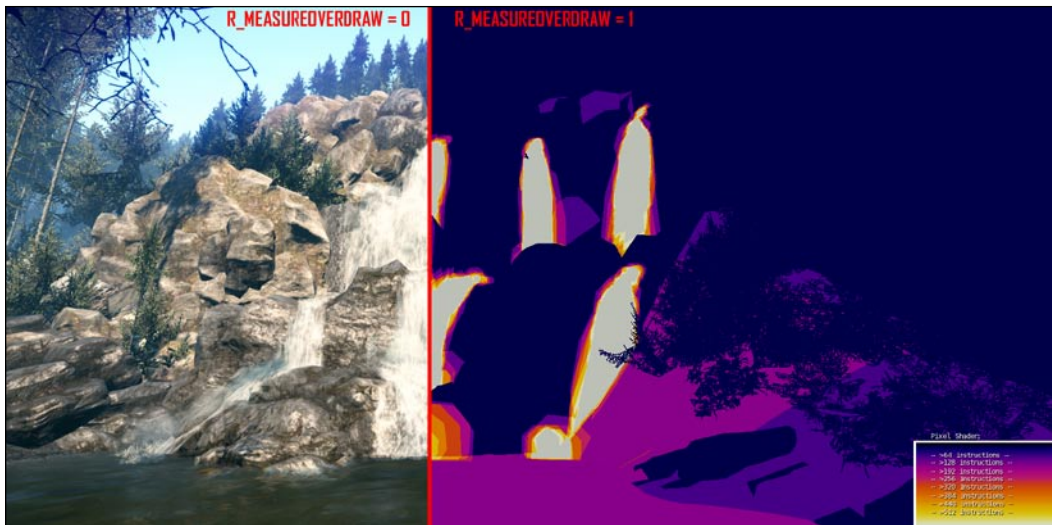


This view allows you to visualize the amount of pixel instructions per pixel in a scene. `Overdraw` is caused by a number of things, including alpha blending where a pixel is calculated multiple times.

In the previous example, you can see the selection of four decals that overlap each other and a road texture that is also alpha blended. The surrounding vegetation and terrain is relatively low cost but the overlapping decals and road add greatly to the pixel cost.

The final section of debug views we will now explore is the entity debug draws. These enable a number of different debug draws that for the ease of writing we will not cover.

Type in `P_draw_helpers 1` to the console. This view is one of the most useful entity debug draws that a developer can use when adjusting physics. This view essentially shows the view the physics system has of the engine. Different levels of physical simulation are shown as different colors in this view:



How it works...

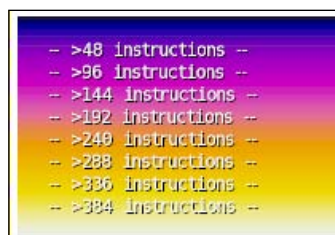
Being able to enable the various debug views is essential! For designers it is especially important as they are the ones that spend the most time in Sandbox putting all the assets together and usually have a direct impact on the overall per view performance in a level.

There's more...

You may want to know more about visualizing overflow in your scene as well as some extra console variables available to you when debugging assets.

Overflow pixel cost scale

Depending on your performance guidelines you may have different budgets or need to view different scales of `r_measureoverflow` command. You can do this by adjusting the `r_MeasureOverflowScale` command, which will adjust the representation color of the number of instructions required in each pixel:



R_stats 15

The `r_stats 15` command is extremely valuable as it displays the frame time cost per render pass. If the value appears red then it is over budget:

GPU Times	
Scene	15.29ms
Shadows	2.36ms
ZPass	1.00ms
Deferred	6.61ms
Cubemaps	0.01ms
Lights	5.32ms
Opaque	3.88ms
Transparent	0.42ms
HDR	0.00ms
PostFX	0.83ms

The goal should be to try to stay below 33ms in the scene property.

Profiles

Profiles can be activated by typing `profile 1` into the console. This enables a per function breakdown cost in frame time. This is especially useful for programmers when scripting and adding new game code.

There are many profiles available and can be explored by using profiles 1-7

See also

- ▶ Go to the *Profiling performance in the Sandbox* recipe earlier in this chapter to learn how to spot issues without having to save a whole printout
- ▶ Go to the next recipe to learn how to optimize levels using VisAreas and portals.

Optimizing the levels with VisAreas and portals

A VisArea is used to define indoor areas that have their own ambient color. You can also create indoor areas without this entity; however you will not be able to achieve totally dark lighting conditions as the bright outdoor ambient lighting also affects indoor areas.

- ▶ Objects inside a VisArea won't be rendered from outside
- ▶ Helps to set up lighting inside rooms
- ▶ With portals you can cut holes inside the VisAreas
- ▶ Portals have to be smaller than the VisArea shape
- ▶ You can enable/disable portals via FG
- ▶ You can have multiple portals in one VisArea

Getting ready

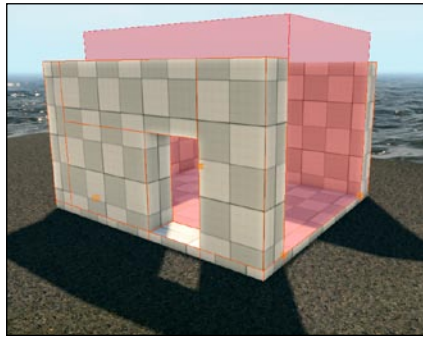
Open any level and set up a small structure similar to the screenshot as shown next.

How to do it...

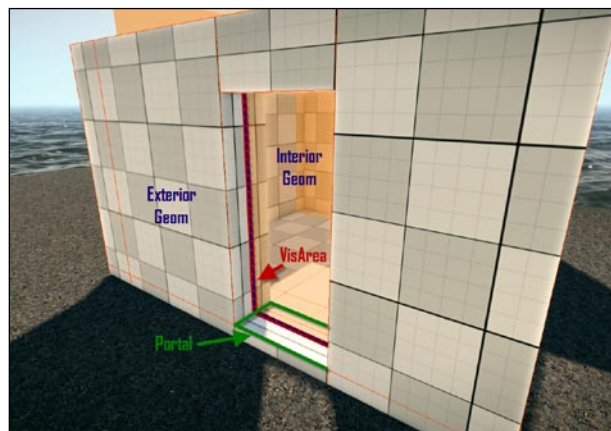
Lets create our own VisArea:

1. On the **RollupBar**, go to **Area | VisArea**.
2. Place the VisArea shape around your room.
3. Set the height.

Be sure everything related is inside the VisArea. Try to stay on the grid. Keep the shape of the VisArea as simple as possible.



4. **Next we must** create a portal for our room. **Portals are used to add a visual entrance** into the VisArea.
5. Create the portal shape according to the size of the entrance:



Keep the size as small as possible (also the size of the portal can't be bigger than the VisArea itself).

6. The portal has to be half inside the VisArea (being in the top and front view helps).
7. If you leave the VisArea (indoor) everything behind you will disappear. including the walls that are located inside.
8. You will need a wall inside the VisArea and a wall outside the VisArea.

How it works...

VisAreas and portals allow for complex effects, like achieving underwater rooms or expansive interiors while rigidly controlling what is being rendered at the time. They are typically used to remove entire building interiors from rendering when the player views them from outside, it also allows you to do the opposite where all exterior rendering is disabled and only the objects within the VisArea are rendered. In either case, more control is given directly to the designer.

There's more...

There are different settings available to you when using portals and VisAreas, and some of these are listed next. You may also want to know how to reduce the blind spots from within the rooms that you have created.

Ambient color of VisAreas and portals

The ambient color parameter specifies which ambient color should be inside the VisArea or portal. This should be thoroughly considered as the color difference between objects in a portal with a different ambient color. The attached VisArea will then cause discontinuity in lighting and visual quality.

Blind spots

You may find when adjusting portals and VisAreas that having any rotation outside of 90 degrees between their areas will cause blind spots and unpredictable behavior. This can be easily fixed by simply ensuring the connection between the portal and VisArea is 90 degrees.

Using VisAreas and portals vertically

To create VisAreas and portals that work vertically, you must create them with their position in mind. They cannot be rotated from vertical to lateral but rather must be drawn lateral instead.

See also

- ▶ Go to the *Making shapes with the Solids tool* recipe in *Chapter 3, Basic Level Layout*, to learn to construct solids
- ▶ Carry onto the next recipe to learn how to use light boxes and light areas within your VisAreas

Using light boxes and light areas

Light clipping boxes and areas are used for the implementation of deferred light clipping in CryENGINE. It is a tool that will enable interior spaces to be lit far better and more accurately than they had been in previous iterations of the CryENGINE.

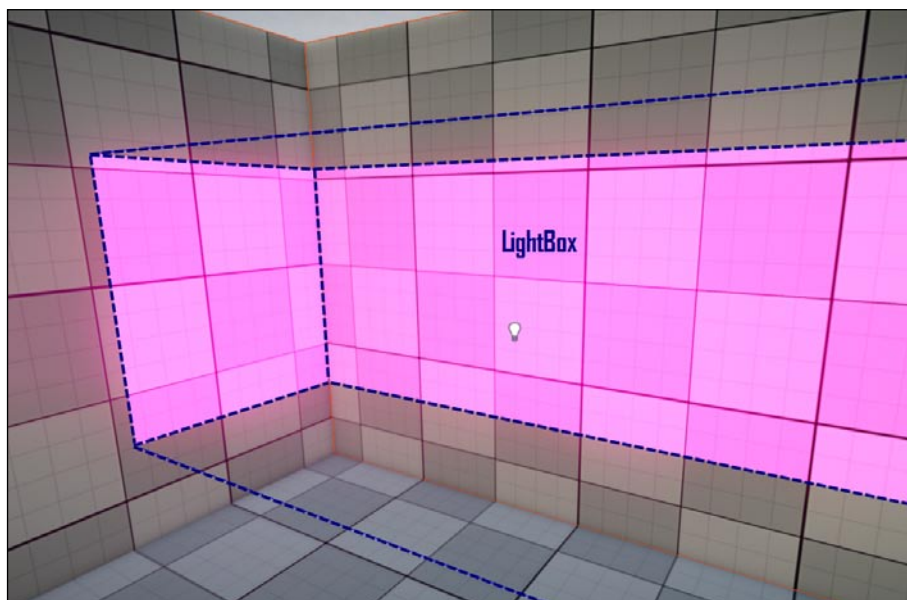
Getting ready

Using light boxes and light areas is highly dependent on the situation. In this example, you should have already completed the prior recipe on *Optimizing the levels with VisAreas and portals*.

How to do it...

Let's test the process of using a light box:

1. Drag-and-drop the light box from the **Area** tab in the **RolloutBar**.
2. Next add a light to your scene.
3. In the lights entity links, add a link to the light box.
4. The light must have the **DeferredClipBounds** property set to **True**. Once set to true, you will see the light will be clipped to that volume.
5. A deferred light that is linked to either a **LightShape** or a **LightBox** will be clipped to that volume regardless of whether it is inside the volume or not:



How it works...

There is a common problem of light bleeding becoming apparent with the deferred approach: the boundaries of lighting are not controllable. Deferred light for example placed in one room can bleed through the wall into another room.

Thankfully, there is a tool available for artists where they can specify a custom stencil culling geometry for each light source in the scene.

This approach is very cheap provided that the clipping geometry is rather coarse and the stencil tagging is very fast on consoles.

There's more...

You may want to know the limits of using light shapes or what occurs when trying to link to multiple light shapes.

Using a concave light shape

If a concave light shape has been created, the editor should warn the user and not clip the light.

Linking to multiple light shapes

If a light falls inside, or is linked to more than one *LightBox* or *LightShape*, the editor will warn the user and clip the light to the first *LightBox*/*LightShape* in the list.

See also

- The *Placing the objects in the world* recipe in *Chapter 2, Sandbox Basics*.

Activating and deactivating the layers

In this recipe, we will explore the use of an important flow graph node that allows designers to stream in and out entire layers of objects.

Getting ready

The requirements for each level will be different and thus to begin, you should have a populated level open that has entities split between different layers.

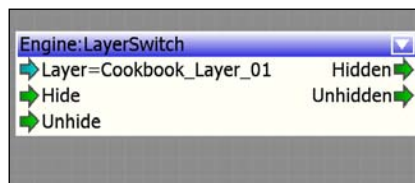
How to do it...

In this example, we will use the layer activation and de-activation to show all the contents required for a particular game play section. Typically, when creating levels, they are split into unique sections called action bubbles. It is common practice to hide layers that contain action bubbles not being played. It is possible to activate and deactivate all entities in a specific layer by using the **Engine: LayerSwitch** flownode.



Even brushes and solids can be hidden/unhidden using this node.

1. Create a new flow graph and add the **Engine: LayerSwitch** flownode to the graph.
2. Next, double-click in the layer property. This **will present you with a pull-down window** allowing you to set the layer that should be attached to this node:



Layer switching needs to be globally enabled in the level in order to work.

3. In the **RollupBar**, switch to the **Terrain** tab, then select **Environment** and set **UseLayersActivation** to **True**, which is in the **EnvState** section.

How it works...

Being able to hide and show different layers, expands the control a designer has on the world while the player is playing in it. It is common that multiple triggers are used to hide certain layers or unhide other layers as the player progresses through a level. This can also save a massive amount of time when it comes to re-working entire areas for a player to re-visit the area after an event.

It is important to know that though it can immediately save performance there is no core manipulation of the streaming system when performing layer switching.

There's more...

You may want to know the limits of layer switching or how to use this tool when it comes to creating a cinematic.

Limits of layer switching

Though this is a simple technique, there are some limits to what it can do:

- ▶ Entities which are set to *Hidden In Game* will not unhide if the *Unhide* input is triggered
- ▶ Picked up objects will disappear in the player's hands when their layer gets hidden
- ▶ Only entities, brushes, and solids are affected
- ▶ A layer which gets hidden/unhidden should never contain AI to avoid conflicts with territories and waves
- ▶ The layer switch has nothing to do with streaming

Cinematics

It is good workflow to always create a layer dedicated to cinematics. The cinematics will typically have to unhide and hide different entities within the world to function, as well as, to be able to maintain a high standard of quality in the cutscene while saving performance on the rest of the level. Using the layer switch combined with the cinematics layer gives a lot of control.

See also

- ▶ Go to the *Debugging the Flow Graph* recipe in *Chapter 9, Game Logic*, to learn how to debug a flow graph

Index

Symbols

- *.cry file** 51
- .anm animations** 197
- .anm files** 196
- .caf animation** 183
- .cdf (Character Definition File)** 170
- .CGA** 163, 199
- .CHR** 163
- .chrparams file**
 - animation names, wildcarding 183
- .cry files** 8
- .pak files**
 - detecting 53
 - opening 52
 - re-exporting 53
- <body> tag** 207
- <Damages> tag** 220
- <DamagesGroups> tag** 220
- <Power> tag** 211
- <SeatAction> tag** 219
- 3ds**
 - material, setting up for export engine 138-142
- 3D Studio Max**
 - CryENGINE 3 plugin, installing 128

A

- Add Event option** 264
- Additonal Animation.** *See* .anm files
- advanced material editor parameters**
 - animate textures, using 154
 - using, for animation creation 153
 - vertex deformation 155
 - working 154

AI

- about 227
- AI_GotoEx 228
- FOVs, narrowing 121
- Goto command, giving 227, 228
- invisible area cross, limiting 113
- invisible boundary cross, limiting 112, 113
- respawning 122-124
- wave FlowGraph node properties 125
- working 228
- AI_Goto** 227
- AI navigation**
 - about 111
 - AI triangulation, generating 111
 - generating 111
 - working 111
- AI triangulation**
 - ai_debugdraw 1 119
 - ai_debugdraw 74 120
 - debugging 119
 - working 120
- Alwave FlowGraph node properties**
 - input 125
 - output 125
- Align Height Map tool** 64
- animation**
 - animation driven motion flag 187
 - assets, types 188
 - creating, advanced material editor parameters used 153
 - filtering 188
 - previewing, for Sandbox 185-187
 - searching 188
 - upper body animation, creating 188-190
 - working 187

ArcadeWheeled parameters 208

Artificial Intelligence. *See* **AI**

AutoMove property 103

B

basic time of day

- Cascaded Shadow Maps 82
- creating 76-79
- Fog subsection 80
- Force sky update to True settings 82
- play icon 83
- record icon 83
- sky color setting 79
- Time Of Day dialog 82
- Variance Shadow Maps 82
- working 82

blendspace 194

Brushes

- Geom entity, differentiating 62

C

camera animation, Track View sequence

- curve editor 250, 251
- Field of View (FOV) 250
- in Track View editor 247, 248
- playback speed 250
- roll, adding to camera 249, 250
- steps 246, 247
- working 250

capture_folder console command 99

capture_frames_once command 99

capture_frames commnad 99

car

- car speed, enhancing 211
- machine gun, attaching 218
- Massbox, manipulating 212
- multiple cameras, setting up 215
- new mesh, creating 199
- new XML, creating 205
- seat helper, creating 213
- weak spot, giving 219

CarDestroy 221

car speed

- Arcade Wheeled movement property 211
- increasing 211

Cascaded Shadow Maps 82

character

- .chrparams file, working 183
- about 163
- animating, from CryENGINE 179
- animation, creating 178
- animation names, wildcarding 184
- animation names, wildcarding within .chrparams file 183
- animations compression, changing 183
- Animobject entity 184
- opening, in character editor 179-183
- previewing, for Sandbox 185-187

character animation

- .chrparams file Wildcard Mapping 183, 184
- animations compression, creating 183
- Animobject entity 184
- creating 178-182
- working 183

cloud feature

- achieving, ways 104
- enhancing 103, 104
- shadows 104
- working 104

color grading

- _CCH naming convention 99
- about 97
- steps 98
- TGA images, using 99
- visual glitches, debugging 99
- working 98

console variables (CVars)

- t_scale cvar 262
- using 260
- using, in Track View 261, 262
- values, animating 262
- working 262

Console Variables window 16

constraints

- about 273
- constraint frame 275
- placing down 274
- properties 275
- working 274

cookbook_wave animation 188

countdown timer

- creating 238, 239

resulting Flow Graph 239
working 239

CryENGINE

animated characters 163
caustics 102
ragdolls 173
skinned characters, creating 164-171

CryENGINE 3

.cry files, working 7
AI 109
asset pipeline 128
cloud feature 102
color grading 98
D, unit setup used 135-137
default settings, restoring 28-30
level, opening 6, 7
level.cfg, using 8
material effects 155
source assets 128
static objects, exporting 143
target Assets 128
user folder, deleting 30

CryENGINE 3 plugin installation, for 3D Studio Max

3ds Max CryTools Maxscripts 130
3ds Max CryTools Maxscripts, installing 130
3ds Max CryTools Maxscripts, uninstalling 130
assets, creating 128
starting with 128, 129
steps 129
tools, classifying 130
working 130

CryENGINE 3 Software Development Kit 5, 31

Crysis:HitInfo node 231

Crytek Geometry Animation. *See* .CGA

CryTIF

about 131
DDS file output, manual generation 135
default presets, adjusting 135
Plug-in Root Path, editing 134
using, for texture creation 131-134
working 134

Ctrl + E 52

Customize dialog box 19

D

debug draw modes

about 290
enabling 290-293
pixel cost scale 293
profiles 294
r_stats 15 command 294
working 293

Decals

parameters 68, 69
using, for terrain tiling breakup 67, 68
working 68

DefaultVehicleDamages.xml 221

DeferredClipBounds property 297

density {{density or =mass}} = mass parameter 152

Density offset parameter 94

Depth Pass 284

destroyable objects

2D breakable assets 151
about 149
creating 149, 151
jointed breakables 152
user defined properties 152
working 151

director node 244

Display Info tab 283

E

Edit Events option 263

Enemy AI

Grunt 110
placing 110
replacing, as Entity Archetypes 110
working 110

engine

.pak files, opening 52
exporting to 52

entity animation, Track View editor

about 254
entity visibility track 257
scale tracking 257
steps 254-256
tracks 257

- working 257
- entity parameter 152**
- environment, creating**
 - basic time of day 76
 - cloud feature 103
 - Color grading 97
 - GI 86
 - global volumetric fog 92
 - HDR lighting 89
 - night scene 95
 - photo realistic ocean 99
 - rain 105
 - terrain lighting 83
- Export Nodes button 196**

F

- FirstPerson class 216**
- First Person view to Third Person view**
 - goc_camera 237
 - switching setup 236, 237
- Flow Graph**
 - about 229
 - debugging 229
 - players health, displaying 234, 235
 - prerequisites 229
 - result, clearing 230
 - sequence, triggering 251
 - working 229, 236
- FlowGraph (FG) 123**
- Follow Terrain method 44**
- Follow Terrain tool 44**
- forbidden areas**
 - cross, limiting 113
 - working 114
- forbidden boundaries**
 - cross, limiting 112
 - working 112
- FOVs, AI**
 - awarenessOfPlayer, working 121
 - FOVPrimary, working 121
 - FOVSecondary, working 122
 - narrowing 121
- FPS 283**
- Frames Per Second. *See* FPS**
- frame time 283**
- Frequency (%) brush 40**

G

- Game Mode**
 - switching to 49, 50
- generic = count parameter 152**
- Geom entity**
 - Brushes, differentiating 62
 - object types 62
 - placing, in level 61
 - working 62
- GI**
 - about 86
 - advanced Cvars 88
 - starting with 87
 - using 87
 - working 88
- Global Illumination. *See* GI**
- global volumetric fog**
 - about 92
 - creating 93, 94
 - Density offset parameter 94
 - fog rendering, disabling in Render Settings 94
 - fog rendering, enabling in Render Settings 94
 - working 94
- Goto command 227**

H

- Hangman on rope example**
 - performing, steps 269-271
 - starting with 269
 - working 272
- HDR Lighting**
 - about 89
 - Environment settings, exploring 89-91
 - flare light effects 91
 - glow texture effect 92
 - working 91

I

- Icon Bar 18**
- image-based lighting**
 - about 159
 - cons 160
 - creating, entity environment probe used 160
 - cubemaps, generating 161
 - cubemaps creating, Material Editor used 161

- pros 160
- working 160

interior object navigation

- AI Navigation Modifier 116
- AI Points 116, 117
- Auto-Dynamic Points vs Designer Controlled Points 118
- Entry/Exit Points 118
- setting up 116
- starting with 114, 115
- working 118

K

Key Properties 264

kill counter

- about 230
- creating 230, 231
- working 231, 232

kill goal

- about 232
- Math:Counter, using 234
- player achievement, rewarding 232-234

L

Launcher 52

- map, running 54

level

- saving 50, 51
- working 51

level.cfg 8

level navigation

- Sandbox Camera 8-10
- Sandbox Camera, using 8-10

level objects

- browsing 25-27
- frozen objects, browsing 28
- hidden objects, browsing 28
- list types 28
- selecting 25-27
- working 28

level optimization, VisArea used 294-296

level statics

- about 285
- Detailed Dependencies tab 289
- physics size 289
- physics tris 289

- render targets 289
- Save Level Statistics, working 289
- saving 285-288
- Textures tab 289

light areas

- about 297
- multiple light shapes, linking to 298
- working 298

light box

- about 297
- concave light shape, using 298
- using 297
- working 298

Live Physics Skeleton. *See* Live Phys Skeleton

Live Phys Skeleton 167

locomotion animations

- 180 degree rotational assets 195
- about 191
- creating 191
- locomotion loops 194
- support structure, creating 191-193
- swimming transitions 194
- vehicle transitions 194
- working 193, 194

LOD (Level of Detail)

- about 172
- creating 172

low gravity

- GravityBox, working 268
- Gravity sphere 269
- setting up 267, 268
- uniform property 269

M

machine gun

- attaching, to car 218
- working 219

Massbox, car

- lighter objects, pushing 212
- working 212

Material Editor

- using, for cubemap creation 161

material effects

- ammo, surface types 159
- creating 155-158
- defining 155

- new surface types, creating 159
- physics block, parameters 159
- working 158

materials

- Physicalize checkbox 142
- setting up, for export engine 138-142
- textures in 3ds Max, assigning 142
- working 142

Math:Equal node 239

Math:Round node 239

menus

- customizing 17
- Keyboard tab 21
- Options tab 20
- personalizing 21
- Reset menu 21
- shortcut key, assigning 21
- working 20

multiple car cameras

- first person camera 216
- setting up 215
- third person camera 216
- wheel camera 217
- working 218

multiple developer collaboration

- external layer limitations 49
- layers, utilizing 47, 48
- working 48

multires 175

N

New button 19

new level

- creating, from scratch 32, 33
- Heightmap resolution 33
- Meters per unit 33
- options, working 33
- Terrain option, using 33
- Terrain size 33

new mesh, car

- creating 199-201
- dummy helpers, using 205
- hierarchy, setting up 201
- limitless possibilities 205
- working 203, 204

night scene

- corona color, adjusting 97
- corona scale, adjusting 97
- creating, time of day parameters used 95, 96
- HDRSetup parameters 97
- moon color, adjusting 97
- SSAO amount 97
- SSAO contrast 97

Night Sky parameter 96

Non-Uniform Scaling 46

Num Sides parameter

- using 57

O

object layers

- activating 299
- cinematics 300
- deactivating 299
- layer switching, limits 300
- requirements 298
- working 299

object placement

- angle snaps 46
- Ctrl + Shift + Click 46
- grids 46
- local direction 46
- refining 44, 45
- Rotation refinement 46

object placing

- about 43
- steps 44
- working 44

objects, grouping

- group, closing 61
- Group tool, working 61
- prerequisites 60
- steps 61

occlusion geometry 148

Ocean Animation parameters 101

Ocean Fog Color 100

P

p_draw_helpers property 273

parameters, Decals

- Deferred 69

- ProjectionType 68
- ViewDistRatio 69
- personalized toolset layout**
 - about 12
 - Console 16, 17
 - Rollup Bar 13
 - setting up 13-15
 - starting with 12
 - Status Bar 16
 - Toolbox 17
 - working 15
- Perspective Viewport window**
 - about 9
 - customizing 10
 - main viewport, splitting to several subviewports 11
 - Speed input 11
 - viewport, adjusting 11
 - working 10
- photo realistic ocean**
 - caustics 102
 - creating 99-101
 - Free form transformation (FFT) water 102
 - water parameters, animating 101
 - working 101
- physics properties, rock slide**
 - ActivateOnDamage 279
 - CanBreakOthers 279
 - Density 279
 - Mass 280
 - Physicalize 280
 - PushableByPlayers 280
 - Resting 280
 - RigidBody 280
 - RigidBodyActive 280
- player camera**
 - changing, key input used 236, 237
 - working 237
- portals**
 - ambient color 296
 - levels, optimizing 294
 - vertical usage 296
 - working 296
- Prefab**
 - benefits 73
 - closing 74
 - creating 72, 73
 - Extract All function 74
 - Extract Object function 74
 - Modified Prefabs 73
 - new library, creating 72
 - opening 74
 - Pick and Attach function 74
 - Remove Object function 74
 - starting with 72
 - Update Prefab function 74
 - working 73
- procedural terrain**
 - generating 34, 35
 - generating, settings 36
 - working 36
- procedural terrain generation setting**
 - Blurring (Blur Passes) 37
 - Bumpiness / Noise (Fade) 36
 - Detail (Passes) 36
 - feature size 36
 - Make Isle 37
 - Set Water Level 37
 - Variation 36
- profiling tools, Sandbox**
 - budget 284, 285
 - Draw Call 284
 - frame time 283
 - Triangle count 284
 - using 282, 283
 - values 282
 - working 283
- properties, constraints**
 - damping 275
 - max_bend_torque 275
 - max_pull_force 275
 - NoSelfCollisions 275
 - Radius 275
 - UseEntityFrame 275
- property, Arcade Wheeled**
 - acceleration 211
 - decceleration 211
 - handbrake-decceleration 212
 - ReverseSpeed 211
 - TopSpeed 211
- Pure Gamemode 52, 53**

R

r_measureoverdraw command 293

r_MeasureOverdrawScale command 293

r_stats 15 command 294

ragdolls, CryENGINE

- about 173, 178
- dead body entity settings 178
- IK limits 177
- ParentFrames 177, 178
- setting up 174-176
- starting with 173
- working 177

rain

- achieving 105
- fog entity 107
- global wind speed, adjusting 107
- lightning entity 106
- particle-only technique 105, 106
- Post Effect entity, working 106

rain.rain.space_loop functions 105

render targets 289

RESOURCE COMPILER 128

rigid body geometry data

- .anm files, working 197
- animating 195
- creating 195, 196, 197
- physically simulated animations, baking into .cga objects 197, 198
- pre-baked .cga, using 198

road parameters

- SortPriority 63
- StepSize 63
- TileLength 63
- width 63

Road tool

- about 62
- Align Height Map tool 64
- road parameters 63
- Shape Editing 64
- starting with 62
- using 62
- working 63

rock slide

- about 277
- building 278, 279

- physics properties 279
- simulation properties 280
- working 279

Rollup Bar

- about 21
- AI section 23
- Archetype entity section 24
- Area section 24
- Entities section 24
- Geom entity section 24
- Misc Objects section 24
- Prefabs section 25
- Solids section 24
- Sound section 25
- starting with 22
- using 22, 23
- working 23

S

Sandbox

- animation, previewing 185-187
- characters, previewing 185-187
- designing 9
- profiling tools 282

Save State button 178

Scale (%) brush 40

seat helper, car

- creating 213, 214
- working 214, 215

Select Objects window 26

sequence trigger, Flow Graph used

- Break on Stop property 254
- debugging trigger, input:key flow node 253
- start time property 253
- steps 251-253
- working 253

set capture_file_format command

- TGA images, using 99

Shape editing

- Adding Points 64
- Angle 64
- Individual Point Width 64

Show Rollup Bar command 20

simulation properties, rock slide

- Damping 280
- FixedDamping 280

- max_time_step 280
- sleep_speed 280

sizevar = var parameter 152

skinned characters, CryENGINE

- about 164
- bone attachments 173
- creating 165-171
- geometry 164
- LODs (Level of Detail), creating 172
- materials, creating 172
- starting with 164
- working 171

Solids tool

- about 56
- basic chimney, creating 59, 60
- basic shapes, creating 56
- cone shape 57
- cylinder shape 57
- editing 57-59
- merging 57-59
- modifying 58
- Num Sides parameter, using 57
- prerequisites 56-58
- selected geometry, exporting to .OBJ 60
- sphere shape 57
- working 57
- XForm, resetting 60

Spawn point, for Pure Gamemode

- creating 53
- working 53

SSAO (screen-space-ambient occlusion) 86

static objects

- about 143
- box parameter 148
- capsule parameter 149
- creating 145
- cylinder parameter 148
- exporting 146, 147
- Mass = Value parameter 149
- models, creating 144
- occlusion geometry 148
- physics proxy 148
- simple cylindrical unwrap 145
- sphere parameter 149
- user defined properties 148
- working 147

T

terrain lighting

- about 83
- adjusting 84
- Moon/Sun Shadow Transition, adjusting 85
- SSAO (screen-space-ambient occlusion) 85, 86
- Terrain Occlusion dialog box 85
- working 85

Terrain sculpting

- about 37
- Noise settings 40
- reposition objects 40
- starting with 37
- using 37, 38, 39
- vegetation 40
- working 39

terrain texture

- Altitude and Slope, setting 43
- Filter (Brightness), setting 43
- Generating Surface Textures, setting 43
- Radius and Hardness, setting 42
- setting up 41, 42
- Tile Resolution, setting 43
- working 42

terraintexture.pak 43

terrain tiling breakup, Decals used

- Decal parameters 68
- starting with 67
- steps 67, 68
- working 68

territory FlowGraph node properties

- input 124
- output 124

textures

- creating, CryTIF used 131

THE COLOR CORRECTION LOOKUP REFERENCE CHART 97

toolbars

- Console ToolBar 18
- customizing 17-20
- Delete button 19
- Dialogs ToolBar 18
- EditMode ToolBar 17
- Mission ToolBar 18

- New button 19
- Object ToolBar 18
- Rename button 19
- Reset button 19
- Standard ToolBar 17
- Terrain ToolBar 18
- working 20

Tornadoes

- about 272
- p_draw_helpers property 273
- placing down 272
- working 272, 273

track events

- image nodes, triggering from 266
- removing, from sequence 266
- using 263-265
- working 265

Track View

- console variables (CVars), using 260-262

Track View editor

- about 241
- AnimObject entities, working 259
- entity, animating 254
- entity animation, playing 258, 259
- loop animation 259
- start time 259
- time scale 260
- track events, using 263
- using 241

Track View sequence

- about 242
- camera, animating 246
- creating 242-244
- director node, capture 246
- director node, console 246
- properties 246
- working 245

trigger area setup

- prerequisites 223
- steps 224-226
- working 227

U

units

- grid setting 137
- measurement references 137

- setting up, for CryENGINE match 135, 136, 137
- snap setting 137
- working 137

upper body only animations

- additive animations 190
- additives, using 190
- creating 188-190
- working 190

V

Variance Shadow Maps 82

vegetation objects, painting

- parameters 66
- starting with 64
- steps 65
- Vegetation Painter, working 66

vegetation parameters

- AlignTo Terrain 66
- Bending 66
- Brightness 66
- CastShadow 66
- Density 66
- ElevationMin/Max 66
- GrowOn Voxels/Brushes 66
- Hideable 66
- Layer_Frozen/Wet 67
- LodDistRatio 67
- Material 67
- MaxViewDistRatio 67
- MinSpec 67
- Pickable 66
- PlayerHideable 66
- RandomRotation 66
- RecvShadow 67
- size 66
- SizeVar 66
- SlopeMin/Max 66
- SpriteDistRatio 67
- Use On Terrain Layers 67
- UseSprites 67
- UseTerrainColor 66

VisArea

- ambient color 296
- blind spots 296
- creating 295, 296

- levels, optimizing 294-296
- vertical usage 296
- working 296

Voxel

- about 69
- caves, making 69-71
- Copy Terrain 72
- enhancing, options 72
- materials 72
- Soft Create 72
- starting with 69
- working 71

W

weak spot

- giving, to car 219, 220
- working 221

wrecking ball

- about 276
- using 276, 277
- working 277

X

XML, for car

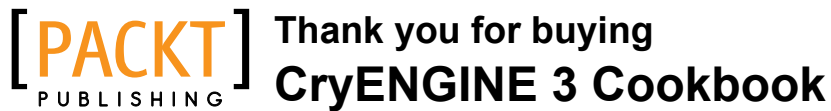
- basic properties 206
- components 210
- creating 206
- def_vehicle.xml file 210
- DefaultVehicle.xml 206
- movement parameters 208, 209
- parts, creating 207, 208
- working 210

Y

y axes 275

Z

Z axis , Shape Editing 64



About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

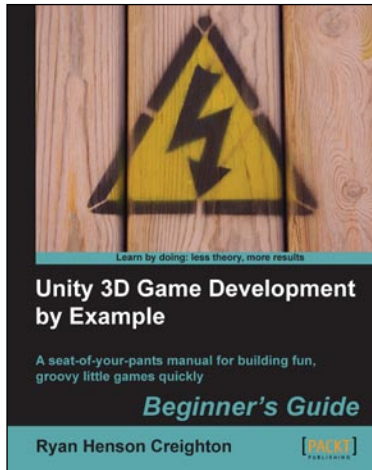
Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

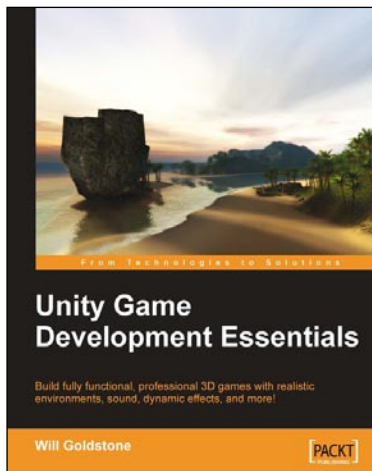


Unity 3D Game Development by Example Beginner's Guide

ISBN: 978-1-849690-54-6 Paperback: 384 pages

A seat-of-your-pants manual for building fun, groovy little games quickly

1. Build fun games using the free Unity 3D game engine even if you've never coded before
2. Learn how to "skin" projects to make totally different games from the same file – more games, less effort!
3. Deploy your games to the Internet so that your friends and family can play them
4. Packed with ideas, inspiration, and advice for your own game design and development



Unity Game Development Essentials

ISBN: 978-1-847198-18-1 Paperback: 316 pages

Build fully functional, professional 3D games with realistic environments, sound, dynamic effects, and more!

1. Kick start game development, and build ready-to-play 3D games with ease
2. Understand key concepts in game design including scripting, physics, instantiation, particle effects, and more
3. Test & optimize your game to perfection with essential tips-and-tricks

Please check www.PacktPub.com for information on our titles