3D Game Development using Unity Engine

Project Report & Documentation

Team Members

Poorva Chodnekar
Shubhangi Gupta
Amey Naiyan
Ganesh Namshikar

Project Guides

Shona Afonso
Rosebud Valaderes

PADRE CONCEICAO COLLEGE OF ENGINEERING

VERNA, GOA - 403722

Department of Computer Engineering

This is to certify that the following students

Roll no.	Name	Seat no.
091107	Poorva Chodnekar	0529
091122	Shubhangi Gupta	0583
091142	Amey Naiyan	0567
091143	Ganesh Namshikar	0568

of the Final Year of Bachelor of Engineering (Computer Engineering), have undertaken and satisfactorily completed the dissertation entitled

"3D Game Development using Unity Engine"

during the 7th and 8th Semesters for the Academic Year, 2012-13.

Dr. Luis Mesquita (Principal)	Dr. Luis Mesquita (Head of Department)	Mrs Shona Afonso (Internal Guide)

PADRE CONCEICAO COLLEGE OF ENGINEERING

VERNA, GOA - 403722

Department of Computer Engineering

3D Game Development using Unity Engine

has been satisfactorily completed by

Roll no.	Name	Seat no.
091107	Poorva Chodnekar	0529
091122	Shubhangi Gupta	0583
091142	Amey Naiyan	0567
091143	Ganesh Namshikar	0568

of the Final Year of Bachelor of Engineering (Computer Engineering).

Dissertation submitted in partial fullfilment for obtaining the degree of
Bachelor of Engineering (Computer Engineering)
during the 7th and 8th Semesters for the Academic Year, 2012-13.

Mrs Shona Afonso	Mrs Rosebud Valaderes	
(Internal Guide)	(Internal Co-Guide)	(External Examiner)

Acknowledgement

It gives us immense pleasure to acknowledge and convey our heartfelt appreciation and affection towards the institution with its management and respected faculty. The task that we undertook would not have evolved successfully without support of some eminent personalities.

First of all we would like to extend to the respected principal and our head of department Dr. Luis Mesquita for his encouragement and words of wisdom. Our heartfelt and sincere thanks to our guides Asst Prof. Shona Afonso and Asst Prof. Rosebud Valaderes who faithfully assisted and supported us In our work. Their sense of motivation and hard work inspired us a lot. A very special thanks to Sunil Hegde for being a great help throughout the development of the project.

We are also grateful to the college for providing the library and laboratory facilities and all the laboratory assistants for their unconditional assistance at all times. We are also obliged to the entire computer science department, our friends, our families and our well wishers for their endorsements during the long and arduous task of fabricating this project.

Above all, the Grace of God led us to finish this project satisfactorily.

Thank you, one and all…

Table of contents

Chapter 1

Video Games

1.1 A Brief History

The history of video games goes as far back as the 1940s, when in 1947 Thomas T. Goldsmith, Jr. and Estle Ray Mann filed a United States patent request for an invention they described as a "cathode ray tube amusement device." Video gaming would not reach mainstream popularity until the 1970s and 1980s, when arcade video games, gaming consoles and home computer games were introduced to the general public.

Late 1950s–1960s

In 1961, a group of students at MIT, including Steve Russell, programmed a game titled Spacewar! on the PDP-1, a new computer at the time. The game pitted two human players against each other, each controlling a spacecraft capable of firing missiles, while a star in the center of the screen created a large hazard for the crafts. In 1966, Ralph Baer engaged co-worker Bill Harrison in the project, They created a simple video game named Chase, the first to display on a standard television set. With the assistance of Baer, Bill Harrison created the light gun.

1970s

In September 1971, Galaxy Game was installed at Stanford University. Based on Spacewar!, this was the first coin-operated video game. Another significant game was Gun Fight, an on-foot, multi-directional shooter, designed by Tomohiro Nishikado and released by Taito in 1975. It depicted game characters, game violence, and human-to-human combat, controlled using dual-stick controls.

Golden age of video arcade games (1978–1986)

The arcade game industry entered its golden age in 1978 with the release of Space Invaders by Taito, a success that inspired dozens of manufacturers to enter the market. The game also became the subject of numerous articles and stories on television and in newspapers and magazines, establishing video gaming as a rapidly growing mainstream hobby. By 1981, the arcade video game industry was generating an annual revenue of $5 billion in North America. Home computers began appearing in the late 1970s and were rapidly evolving in the 1980s, allowing their owners to program simple games. Hobbyist groups for the new computers soon formed and PC game software followed.

Third generation consoles (1983–1995) (8-bit)

In 1985, the American Video Game Console market was revived with Nintendo’s release of the Nintendo Entertainment System (NES). The NES instantly became a success, dominating the North American and Japanese home console gaming markets until the rise of the next generation of 16-bit consoles in the early 1990s.

Fifth generation consoles (1993–2006) (32 and 64-bit)

In 1994, three new consoles were released in Japan: the Sega Saturn, the Sony PlayStation, and the PC-FX, the Saturn and the PlayStation later seeing release in North America in 1995. The PlayStation quickly outsold all of its competitors mainly on the strength of its available titles.

Mobile phone gaming

Mobile phones began becoming video gaming platforms when Nokia installed Snake onto its line of mobile phones in 1997 (Nokia 6110). Soon every major phone brand offered "time killer games" that could be played in very short moments such as waiting for a bus.

1.2 Video Game Genres

Video game genres are used to categorize video games based on their gameplay interaction rather than visual or narrative differences. A video game genre is defined by a set of gameplay challenges. They are classified independent of their setting or game-world content, unlike other works of fiction such as films or books. Following is a listing of commonly used video game genres with brief descriptions and examples of each. This list is by no means complete or comprehensive.

1.2.1 Action

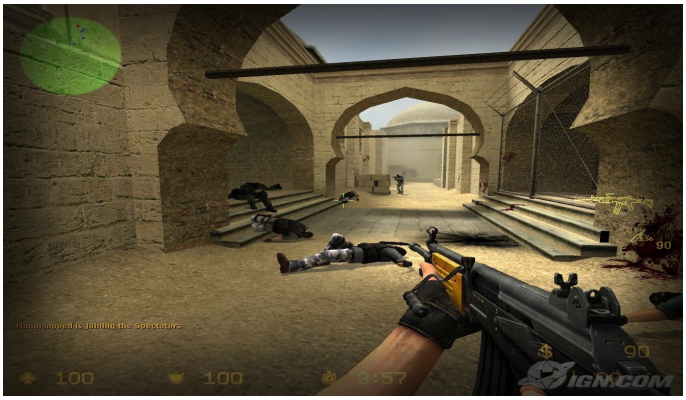
An action game requires players to use quick reflexes, accuracy, and timing to overcome obstacles. There are many subgenres of action games, such as fighting games and first-person shooters.

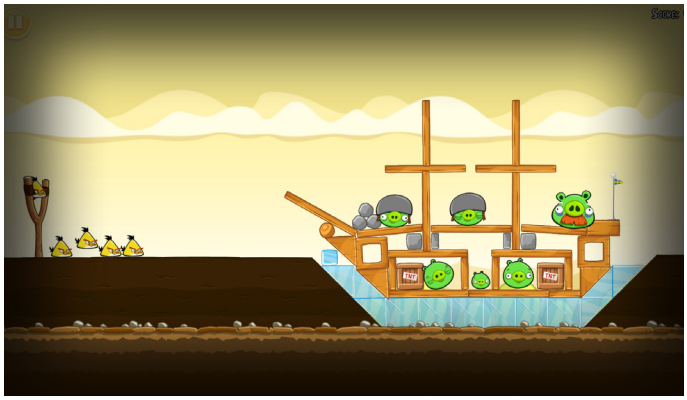
Ball and paddle : The predecessor of all console game genres, a ball-and-paddle game was the first game implemented on a home console (Pong). Later renditions have included Breakout, which was a driving influence behind the Apple II computer

Beat 'em up and hack and slash : Have an emphasis on one-on-many close quarters combat, beating large numbers of computer-controlled enemies. Beat 'em ups feature hand-to-hand combat, and hack and slash games feature melee weaponry, particularly bladed weapons. Both genres feature little to no use of firearms or projectile combat.

First-person shooter : First-person shooter video games, commonly known as FPSs, emphasize shooting and combat from the perspective of the character controlled by the player. This perspective gives the player the feeling of "being there", and allows the player to focus on aiming. Most FPSs are very fast-paced and require quick reflexes on high difficulty levels.

Third-person shooter : Third-person shooter video games, known as TPSs or 3PSs, emphasize shooting and combat from a camera perspective in which the player character is seen at a distance. This perspective gives the player a wider view of their surroundings as opposed to the limited viewpoint of first-person shooters. Furthermore, third-person shooters allow for more elaborate movement such as rolling or diving, as opposed to simple jumping and crouching common in FPS games.





1.2.2 Arcade

Arcade games is a genre often used to describe a vast variety of games ranging from Angry Birds (could also be classified as a physics-based game) to Pac Man to other games that differ drastically in game play. Obviously, the games that came out on the arcade machines when they were popular are commonly classified and refereed to as arcade games. Games that are classified as arcade usually have a score system, little (like Angry Birds) or no story (like Pac Man), have linear game progression, have obvious and well-defined levels (like Pac Man and Angry Birds), etc

1.2.3 Role Playing Games

Most cast the player in the role of one or more "adventurers" who specialize in specific skill sets (such as melee combat or casting magic spells) while progressing through a predetermined storyline. Many involve maneuvering these character(s) through an overworld, usually populated with monsters, that allows access to more important game locations, such as towns, dungeons, and castles. Gameplay elements strongly associated with RPGs, such as statistical character development through the acquisition of experience points, have been widely adapted to other genres such as action-adventure games.

1.2.4 Simulation

Simulation video games is a diverse super-category of games, generally designed to closely simulate aspects of a real or fictional reality.
Construction and management simulation : In city-building games the player acts as overall planner or leader to meet the needs and wants of game characters by initiating structures for food, shelter, health, spiritual care, economic growth, etc. Success is achieved when the city budget makes a growing profit and citizens experience an upgraded lifestyle in housing, health, and goods. While military development is often included, the emphasis is on economic strength.
Life simulation : involve living or controlling one or more artificial lives. A life simulation game can revolve around individuals and relationships, or it could be a simulation of an ecosystem.
Vehicle simulation : Vehicle simulation games are a genre of video games which attempt to provide the player with a realistic interpretation of operating various kinds of vehicles.

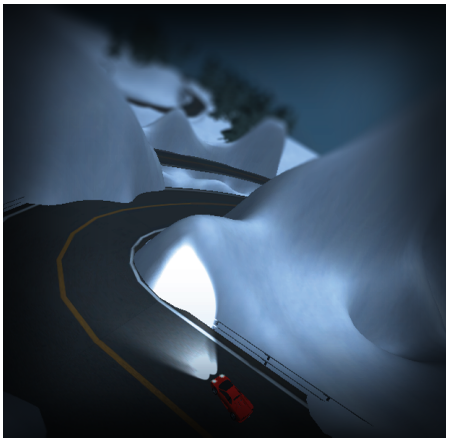
1.2.5 Strategy

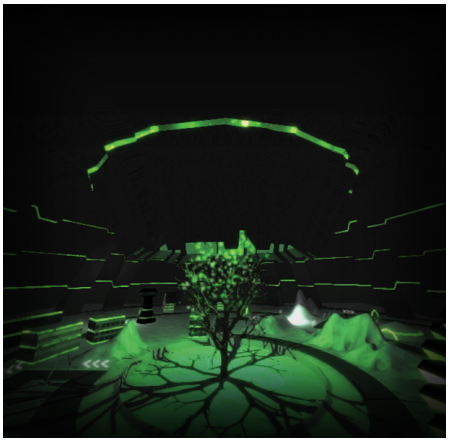
Strategy video games focus on gameplay requiring careful and skillful thinking and planning in order to achieve victory. Strategy video games generally take one of four archetypal forms, depending on whether the game is turn-based or real-time and whether the game's focus is upon strategy or military tactics. **Real-time strategy (RTS)** : This genre is probably the most well known of strategy games and is what most websites mean when they say "strategy games". RTS gameplay is characterised by obtaining resources, building bases, researching technologies and producing units.
Turn-based strategy : A player of a turn-based game is allowed a period of analysis before committing to a game action, and some games allow a certain number of moves or actions to take place in a turn.

1.2.6 Sports

Sports games emulate the playing of traditional physical sports. Some emphasize actually playing the sport, while others emphasize the strategy behind the sport (such as Championship Manager). Others satirize the sport for comic effect (such as Arch Rivals). One of the best selling series in this genre is the FIFA (video game series) series. This genre emerged early in the history of video games (e.g., Pong) and remains popular today.

1.3 Velocé





Veloce is our try at making a video game. Its the result of approximately 10 months of consistent and dedicated work at creating something out of the box. In a nutshell, Veloce is an Arcade-style Racing game. A racing game is a form of video game in which the player controls a vehicle to complete certain tasks or objectives. The player could compete either against a clock, a computer driven vehicle or even another player. There are basically two genres of racing games:-

1) Arcade
2) Simulation

Simulation racing games involve acurate physics. Usually the handling and power specifications of vehicles accurately resemble their real world counteparts. In deep contrast, Arcade style Racing games are designed specifically for an experience that is exciting as well as minimally frustrating.

1.3.1 Aim

Our major goal was to create a video game that is fast, exciting as well as having a smooth learning curve. It is also designed to be simple and consists of just three modes of gameplay.

Our objective was to make a software that would act as a stepping stone for us into the world of Video game development. Through our project we would get an insight into the process of making a Three Dimensional Video game and the use of a game engine like Unity to accelerate the development of a video game enabling us to make better and more advanced games in the future.

1.3.2 Motivation

Video games have always been part of almost every individual in this planet who were fortunate enough to experince or own these virtual wonders. Right from childhood we have been experiencing and often mastering many video games in various different forms, right from Super Mario Brothers on the Nintendo NES to Pokemon Yellow on the Gameboy Color to more recent games like Call of Duty and Need for Speed on the PC and other Gaming consoles.

While the thrill of playing a game always existed, there was always an enigma as to how these games were actually made. Some of us have even tried to make our own little versions of video games with whatever miniscule knowledge we had about the computer and its working. When we were given an option to select our project, we knew we had to create something interesting and fun and ultimately the choice was clear.

Another aspect that came into play was Nostalgia. If you ask any 20 year old individual to name his "Most favorite game of all time". Chances are that he might name a game from the 90's. Well thats how life works, our brain tends to prioritize things that matter to us the most. But there was something about the old games that made us want to play it over and over again. Older games have a certain something, a soul that newer games lack in. It could be maybe because of the shear difficulty due to which one had to put in hours of effort to complete a single level. But at the end all this doesnt matter cause the absolute satisfaction of clearing a level or unlocking a new map or level completely eclipses the frustration of constantly losing. In short it puts a big smile on your face and permanently embeds this feeling into you brain.

Thats exactly what we wanted to build, a game that would bring a smile onto a persons face, it may not be forgiving at first, but once u get past the initial phase, its simple, fast and most importantly fun. Our motive was not technological advancement, it was just about going back to the roots and finding out just what made those games so special.

1.3.3 Overview

Veloce is an Italian word which means "Fast" in English. Build from ground up, every 3D model in this game was designed by us from scratch and it uses the Unity Game development engine to provide the necessary physics simulation and rendering for the game. Prior to the development of this game, we had absolutely no experience or exposure as to how we shall proceed to take our very first steps. We knew we had a mighty task ahead of us and a very prominent deadline. We needed something that would enable us to concentrate on the actual game design and not worry too much about the internal optimizations and physics emulation. Thats where we came across Unity.

The Unity Game Development engine can be described as a set of tools that provides the necessary platform to design and script a game using a very user friendly interface that incorporates drag and drop capabilities. There are other game engines as well like the Cry Engine by Crytek, or the Unreal Engine by Epic Games, or the Anvil Engine by Ubisoft and many more. We chose unity, firstly because it was simple, it had some very powerful tools but packaged in a very user friendly manner, secondly because of the well written documentation that was not only easy to understand, but also complete, and lastly because of the huge active developer forum that has answers to almost any problems that might arise in our development process

1.3.4 Hardware and Software Requirements

Minimum Hardware Requirements include:-
- 1 GHz Pentium Processor or higher
- 1 GB RAM
- 3D Accelerated Video Card supporting DirectX 9.0c and Shader model 3.0
- 1 GB of Hard drive space

Software Requirements include:-
- Windows 8 / 7 / Vista / XP
- DirectX 9.0c

1.3.5 Walkthrough

When you start the Application, you will notice the Unity logo and the splash screen with animation. You will then arrive at the User Profiles page. The game supports upto 4 user profiles indicated by the 4 slots. Initially all slots are empty. You have to create a new profile in any of the slots to proceed to the next scene.

Once you have created the User profile successfully, you will be greeted by the Main Menu. The Menu Options are placed on a rotating carousel that can be controlled using the mouse motion. It consists of 5 Buttons : Single Player, Multi Player, Options, Credits and Quit.

When you press Single Player, two additional options popup: Quick Race and Time Trial. In the Quick Race Mode, you will be competing against 3 other computer driven cars, In the time trial mode you will be competing against the clock. Clicking on either option, you will be taken to the track selection menu, Initially only the first track is unlocked. Subsequent tracks will be unlocked after clearing each level in the Quick Race mode. You can select the number of laps you desire to race.

When you press Multiplayer, 1 additional option will popup ie Split-Screen. In Spit Screen mode you can compete against another player on the same machine itself. Additional modes can be added in future releases.

After selecting a track you will be taken to a car selection menu. Initially only the first car is unlocked. Subsequent cars will be unlocked after clearing each level in the Time Trial mode. In the Split Screen Mode, the car selection menu will allow both players to select cars. Players also have the option to choose a unique color for their car.

Once you successfully select a car and a track, Its now time to race!. Your car will be positioned on the start-finish line. After a Countdown animation, your Race begins!. Depending on which mode you have selected, you may have to race against the computer, against a clock or against your own friends.

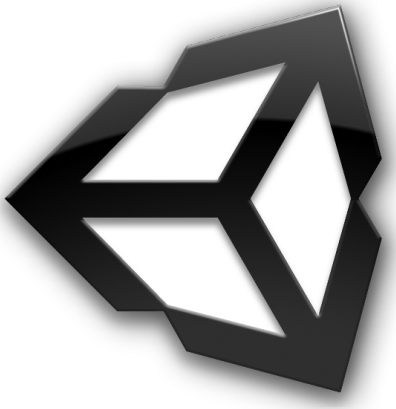
After the race has ended, you will be directed to a result page that shows your result and also notifies you, if you have unlocked any new vehicle or car. You will have the option to restart the race or Go back to the main menu.

You can Exit the game to the desktop by selecting the Quit Option in the main menu.

Chapter 2

Softwares

2.1 Unity Engine 3.4.0 Pro



2.1.1 Overview

Unity (also called Unity3D) is a cross-platform game engine with a built-in IDE developed by Unity Technologies. It is used to develop video games for web plugins, desktop platforms, consoles and mobile devices, and is utilized by over one million developers. Unity is primarily used to create mobile and web games, but can also deploy games to consoles or the PC. The game engine was developed in C/C++, and is able to support code written in C#, JavaScript or Boo. Unity is a game development ecosystem: a powerful rendering engine fully integrated with a complete set of intuitive tools and rapid workflows to create interactive 3D content; easy multiplatform publishing; thousands of quality, ready-made assets in the Asset Store and a knowledge-sharing Community.
Unity supports deployment to multiple platforms. Unity lets you target all platforms and switch between them from a single tool. Within a project you have control over delivery to all platforms including mobiles, web, desktops, and consoles. Unity makes it simple to keep your code working across many devices by abstracting away the majority of platform differences while maintaining the option for precise control when needed. Unity also allows specification of texture compression and resolution settings for each platform a game supports. This means a single high resolution file will work for all targets. By taking the pain out of the development process and doing all of the background work, Unity allows developers to focus on making games.

2.1.2 Unity Interface

The Main Editor Window is made up of several Tabbed Windows, called Views. There are several types of Views in Unity - they all have specific purposes which are described in the subsections below.

Project Browser
The left panel of the browser shows the folder structure of the project as a hierarchical list. When a folder is selected from the list by clicking, its contents will be shown in the panel to the right. The individual assets are shown as icons that indicate their type (script, material, sub-folder, etc). Above the project structure list is a Favorites section where you can keep frequently-used items for easy access. You can drag items from the project structure list to the Favourites and also save search queries there. Just above the panel is a "breadcrumb trail" that shows the path to the folder currently being viewed. The separate elements of the trail can be clicked for easy navigation around the folder hierarchy.

Hierarchy
The Hierarchy contains every GameObject in the current Scene. Some of these are direct instances of asset files like 3D models, and others are instances of Prefabs, custom objects that will make up much of your game. You can select objects in the Hierarchy and drag one object onto another to make use of Parenting . As objects are added and removed in the scene, they will appear and disappear from the Hierarchy as well.
Parenting: Unity uses a concept called Parenting. To make any GameObject the child of another, drag the desired child onto the desired parent in the Hierarchy. A child will inherit the movement and rotation of its parent. You can use a parent object's foldout arrow to show or hide its children as necessary.

Toolbar
The Toolbar consists of five basic controls. Each relate to different parts of the Editor.

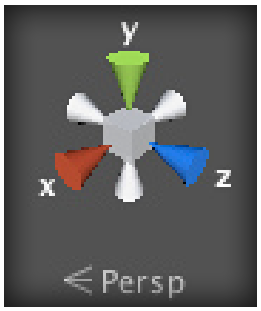

Transform Tools -- used with the Scene View.

Transform Gizmo Toggles -- affect the Scene View display


Play/Pause/Step Buttons -- used with the Game View

Layers Drop-down -- controls which objects are displayed in Scene View.

Layout Drop-down -- controls arrangement of all Views

Scene View


The Scene View is your interactive sandbox. You will use the Scene View to select and position environments, the player, the camera, enemies, and all other GameObjects. Maneuvering and manipulating objects within the Scene View are some of the most important functions in Unity, so it's important to be able to do them quickly. To this end, Unity provides keystrokes for the most common operations.

In the upper-right corner of the Scene View is the Scene Gizmo. This displays the Scene Camera's current orientation, and allows you to quickly modify the viewing angle. Each of the coloured "arms" of the gizmo represents a geometric axis. You can click on any of the arms to set the camera to an orthographic (i.e., perspective-free) view looking along the corresponding axis. You can click on the text underneath the gizmo to switch between the normal perspective view and an isometric view. While in isometric mode, you can right-click drag to orbit, and Alt-click drag to pan.

Inspector
Games in Unity are made up of multiple GameObjects that contain meshes, scripts, sounds, or other graphical elements like Lights. The Inspector displays detailed information about your currently selected GameObject, including all attached Components and their properties. Here, you modify the functionality of GameObjects in your scene.
Any property that is displayed in the Inspector can be directly modified. Even script variables can be changed without modifying the script itself. You can use the Inspector to change variables at runtime to experiment and find the magic gameplay for your game.In a script, if you define a public variable of an object type (like GameObject or Transform), you can drag and drop a GameObject or Prefab into the Inspector to make the assignment. If you have a Prefab selected, some additional buttons will be available in the Inspector.

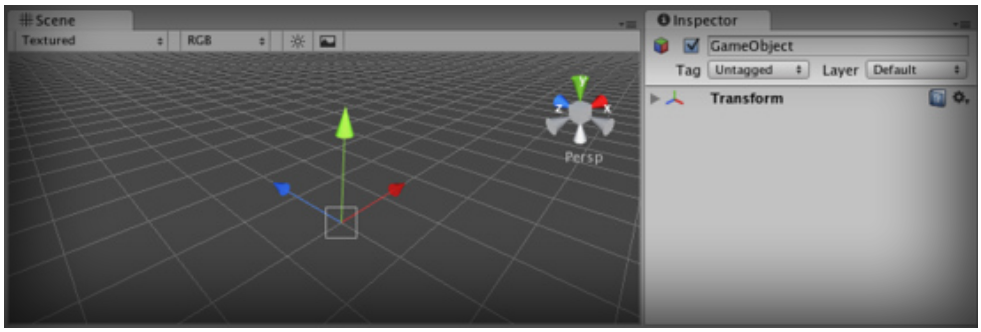
Unity allows assets to be marked with Labels to make them easier to locate and categorise. The bottom item on the inspector is the Asset Labels panel. You can select one or more items from the labels menu to mark the asset with those labels (they will also appear in the Labels panel). If you click a second time on one of the active labels, it will be removed from the asset. The menu also has a text box that you can use to specify a search filter for the labels in the menu. If you type a label name that does not yet exist and press return/enter, the new label will be added to the list and applied to the selected asset. If you remove a custom label from all assets in the project, it will disappear from the list.Once you have applied labels to your assets, you can use them to refine searches in the Project Browser.

Other Views
The Views described on this page covers the basics of the interface in Unity. The other Views in Unity are described elsewhere on separate pages:
•The **Console** shows logs of messages, warnings, and errors.
•The **Animation View** can be used to animate objects in the scene.
•The **Profiler** can be used to investigate and find the performance bottle-necks in your game.
•The **Asset Server View** is used to manage version control using Unity's Asset Server.
•The **Lightmapping View** can be used to manage lightmaps using Unity's built-in lightmapping.
•The **Occlusion Culling View** is used to manage Occlusion Culling for improved performance.

2.1.3 Game Objects

GameObjects are containers for all other Components. All objects in your game are inherently GameObjects. Technically you can create a Component without GameObject, but you won't be able to use it until you apply it to a GameObject.

Empty Game Object
If you want to create a Component from script, you should create and empty GameObject and then add required Component using gameObject.



AddComponent(ClassName) function. You can't create Component and then make a reference from object to it.
From scripts, Components can easily communicate with each other through message sending or the GetComponent(TypeName) function. This allows you to write small, reusable scripts that can be attached to multiple GameObjects and reused for different purposes.

Text-GUI
GUI Text displays text of any font you import in screen coordinates. GUI Texts are used to print text onto the screen in 2D. The Camera has to have a GUI Layer attached in order to render the text. Cameras include a GUI Layer by default, so don't remove it if you want to display a GUI Text. GUI Texts are positioned using only the X and Y axes. Rather than being positioned in World Coordinates, GUI Texts are positioned in Screen Coordinates, where (0,0) is the bottom-left and (1,1) is the top-right corner of the screen. By default, GUI Texts are rendered with Pixel Correct enabled. This makes them look crisp and they will stay the same size in pixels independent of the screen resolution.

Primitive Objects

Unity Game object consists of following primitive objects:

- •Cube
- •Sphere
- •Capsule
- •Cylinder
- •Plane

In order to create a primitive GameObject click GameObject > Create Other > GameObject (e.g.Cube)in the menus to add a primitive to the Scene. A GameObject appears in the scene. The GameObject will be contained in the Prefab. To create the Prefab, click the Create button in the Project panel, and choose Prefab from the resulting menu. Click and drag the Cube GameObject from the Hierarchy panel into the empty grey Prefab in the Project panel. The Prefab lights up blue to indicate that it's got junk in the trunk.

Light

Lights will bring personality and flavor to your game. You use lights to illuminate the scenes and objects to create the perfect visual mood. Lights can be used to simulate the sun, burning match light, flashlights, gun-fire, or explosions, just to name a few.

There are four types of lights in Unity:

•Directional lights are placed infinitely far away and affect everything in the scene, like the sun.

•Point lights shine from a location equally in all directions, like a light bulb.

•Spot lights shine from a point in a direction and only illuminate objects within a cone - like the headlights of a car.

You can create a texture that contains an alpha channel and assign it to the Cookie variable of the light. The Cookie will be projected from the light. The Cookie's alpha mask modulates the light amount, creating light and dark spots on surfaces. They are a great way af adding lots of complexity or atmosphere to a scene.

In Unity Pro all Lights can optionally cast Shadows. This is done by selecting either Hard Shadows or Soft Shadows for the Shadow Type property of each individual Light

Camera

Cameras are the devices that capture and display the world to the player. By customizing and manipulating cameras, you can make the presentation of your game truly unique. You can have an unlimited number of cameras in a scene. They can be set to render in any order, at any place on the screen, or only certain parts of the screen. Cameras are essential for displaying your game to the player. They can be customized, scripted, or parented to achieve just about any kind of effect imaginable.

You can create multiple Cameras and assign each one to a different Depth. Cameras are drawn from low Depth to high Depth. In other words, a Camera with a Depth of 2 will be drawn on top of a Camera with a depth of 1. You can adjust the values of the Normalized View Port Rectangle property to resize and position the Camera's view onscreen. This can create multiple mini-views like missile cams, map views, rear-view mirrors, etc.

2.1.4 Components

Rigidbody

To put an object under physics control, simply add a Rigidbody to it. When you do this, the object will be affected by gravity, and can collide with other objects in the world.

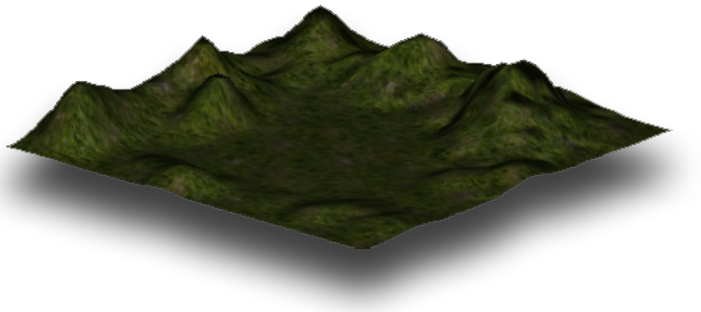
Rigidbodies are physically simulated objects. You use Rigidbodies for things that the player can push around, for example crates or loose objects, or you can move Rigidbodies around directly by adding forces to it by scripting.

If you move the Transform of a non-Kinematic Rigidbody directly it may not collide correctly with other objects. Instead you should move a Rigidbody by applying forces and torque to it. You also use Rigidbodies to bring vehicles to life, for example you can make cars using colliders. Colliders work with Rigidbodies to bring physics in Unity to life. Whereas Rigidbodies allow objects to be controlled by physics, Colliders allow objects to collide with each other. Colliders must be added to objects independently of Rigidbodies. A Collider does not necessarily need a Rigidbody attached, but a Rigidbody must be attached in order for the object to move as a result of collisions.

Colliders
Box Collider - is a basic cube-shaped collision primitive. The Box Collider can be resized into different shapes of rectangular prisms. It works great for doors, walls, platforms, etc. It is also effective as a human torso in a ragdoll or as a car hull in a vehicle. Of course, it works perfectly for just boxes and crates as well!
Mesh Collider - takes the graphical mesh and uses it as a collision shape. The Mesh Collider takes a Mesh Asset and builds its Collider based on that mesh. It is far more accurate for collision detection than using primitives for complicated meshes. Mesh Colliders that are marked as Convex can collide with other Mesh Colliders. It builds its collision representation from the Mesh attached to the GameObject, and reads the properties of the attached Transform to set its position and scale correctly.
Wheel Collider - A special collider for grounded vehicles(example wheels of a car). It has built-in collision detection, wheel physics, and a slip-based tire friction model. It can be used for objects other than wheels, but it is specifically designed for vehicles with wheels.

2.1.5 Unity terrains

Unity provides refined in-editor tools to carve, raise, and lower sweeping and mountainous terrains. It uses a variety of brushes to paint trees, bushes, rocks, grass and other elements into the landscape which makes landscapes look lively and realistic, without compromising on performance.



Creating a new Terrain
A new Terrain can be created from Terrain->Create Terrain. This will add a Terrain to your Project and Hierarchy Views. If a differently sized Terrain is required, choose Terrain->Set Resolution from the menu bar.
After creating the new terrain the following values can be changed.

- **Terrain Width**: The width of the Terrain in units.
- **Terrain Height**: The height of the Terrain in units.
- **Terrain Length**: The length of the Terrain in units.
- **HeightMap Resolution**: The HeightMap resolution for the selected Terrain.
- **Detail Resolution**: The resolution of the map that controls grass and detail meshes. For performance reasons (to save on draw calls) the lower you set this number the better.
- **Control Texture Resolution**: The resolution of the splat map used to layer the different textures painted onto the Terrain.
- **Base Texture Resolution**: The resolution of the composite texture that is used in place of the splat map at certain distances.

Navigating the Terrain

Terrains work a bit differently than other GameObjects. You can use Brushes to paint and manipulate your Terrain. If you want to reposition a Terrain, you can modify its Transform Position values in the Inspector. This allows you to move your Terrain around, but you cannot rotate or scale it.

While your Terrain is selected in the Hierarchy, you can gracefully navigate the terrain with the F (focus) key. When you press F, wherever your mouse is positioned will be moved to the center of the Scene View. This allows you to touch up an area, and quickly zoom over to a different area and change something else. If your mouse is not hovering over an area of the Terrain when you press the F key, the entire Terrain will be centered in your Scene View.

Editing the Terrain

With the Terrain selected, you can look at the Inspector to see some incredible new Terrain editing tools. Each rectangular button is a different Terrain tool. There are tools to change the height, paint splat maps, or attach details like trees or rocks. To use a specific tool, click on it. You will then see a short description of the tool appear in text below the tool buttons.

Most of the tools make use of a brush. Many different brushes are displayed for any tool that uses a brush. To select a brush, just click on it. The currently selected brush will display a preview when you hover the mouse over the terrain, at the size you have specified.

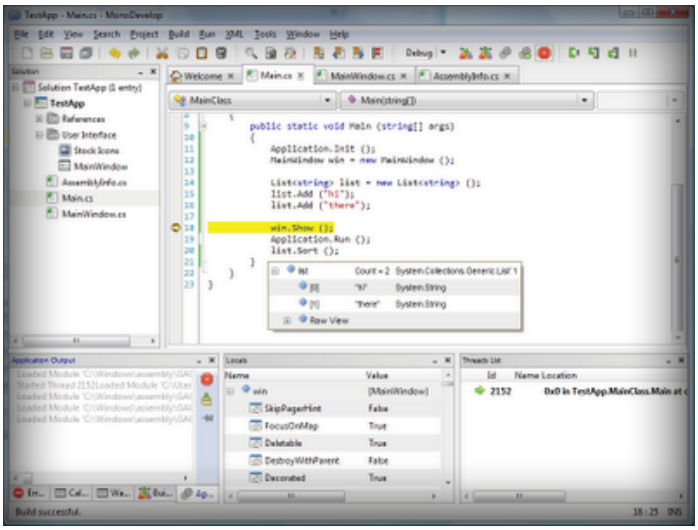
These brushes can be used in the Scene View to paint directly onto the Terrain. Simply choose the tool and brush , then click & drag on the Terrain to alter it in real-time. To paint height, textures, or decorations, the Terrain must be selected in the Hierarchy View.

2.1.6 MonoDevelop

Unity game engine's scripting is built on Mono, the open-source implementation of the .NET Framework. Programmers can use UnityScript (a custom language with ECMAScript-inspired syntax), C# or JavaScript or Boo (which has a Python-inspired syntax). Starting with the 3.0 release, Unity ships with a customized version of MonoDevelop for debugging scripts.

MonoDevelop is an open source integrated development environment for the Linux platform, Mac OS X and Microsoft Windows, primarily targeted for the development of software that uses both the Mono and Microsoft .NET frameworks. MonoDevelop integrates features similar to those of NetBeans and Microsoft Visual Studio, such as automatic code completion, source control, a graphical user interface (GUI) and Web designer. MonoDevelop integrates a Gtk# GUI designer called Stetic. It currently has language support for C#, F#, Java, Boo, Visual Basic.NET, Oxygene, CIL, Python, Vala, C and C++. The big advantage of using Mono over something like Unitron is the code-hinting, which will make writing the code much faster and more efficient.

If a script is double-clicked in the project tab, Monodevelop is the application that loads and edits/debugs it.



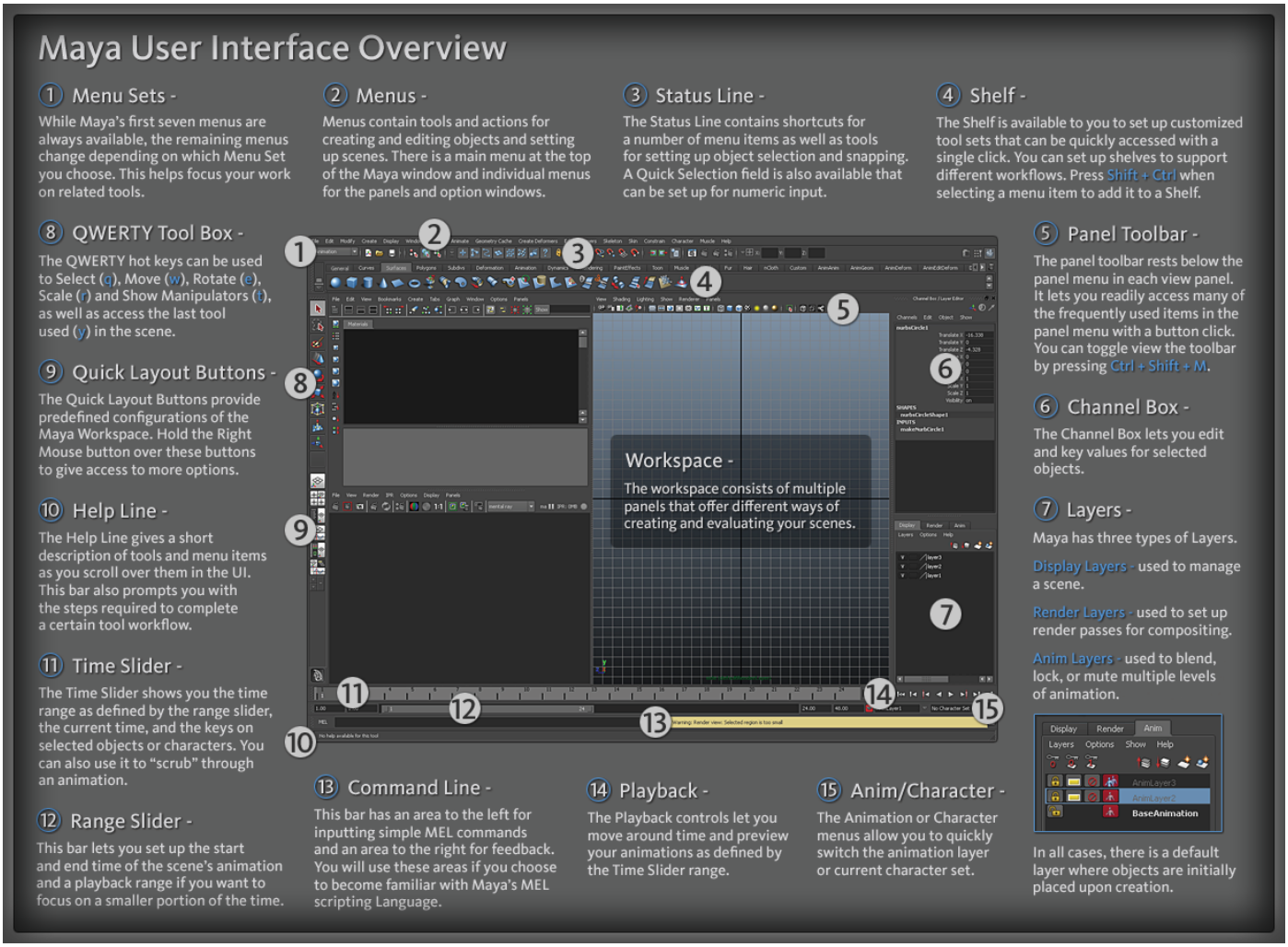
2.2 Autodesk Maya 2012

2.2.1 Overview

Autodesk Maya,commonly shortened to Maya, is 3D computer graphics software that runs on Windows, Mac OS and Linux, originally developed by Alias Systems Corporation and currently owned and developed by Autodesk, Inc. It is used to create interactive 3D applications, including video games, animated film, TV series, or visual effects. The product is named after the Sanskrit word Maya, the Hindu concept of illusion.
Autodesk Maya 3D animation software delivers a comprehensive creative feature set with tools for animation, modeling, simulation, rendering, match moving, and compositing on a highly extensible production platform. For visual effects, game development, post production, or other 3D animation projects, Maya offers toolsets to help meet demanding production requirements.

2.2.2 Maya Interface

The Maya workspace



The Maya workspace is the space where most of the work within Maya is conducted. The workspace is the central window where the objects and most editor panels appear.
When you start Maya for the first time, the workspace displays by default in a perspective window, or panel.

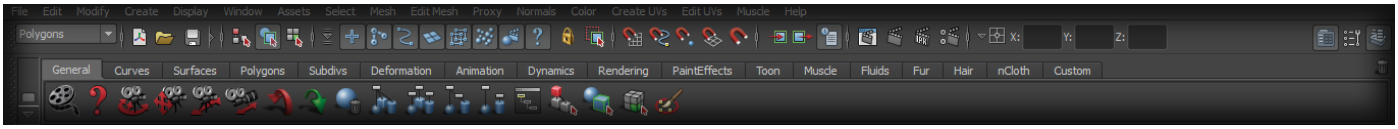
There are the other components of the default perspective view panel:

- The panel is labeled persp at the bottom to indicate that you are viewing the Maya scene from a perspective camera view.
- The panel has its own menu bar at the top left corner of the panel. These menus allow you to access tools and functions related to that specific panel.
- The grid is displayed with two heavy lines intersecting at the center of the Maya scene.

This central location is called the origin. The origin is the center of Maya's 3D world, and with all object's directional values measured from this location.

In Maya, like many other 3D applications, the three dimensions are labeled as the X, Y, and Z axes. The origin is located at X, Y, Z position of 0, 0, 0. The grid also lies along the X, Z plane. We refer to this as a plane because you might visualize an imaginary, flat, two-dimensional square laying along this 3D position.
Maya labels the X, Y, and Z axes with a color scheme: red for X, green for Y, and blue for Z. Many tools that you use in Maya use this color scheme to indicate that you are accessing a particular item that relates to X, Y, and Z in some way. The axis indicator shows in which direction, X, Y, or Z, you are viewing the Maya scene. The axis indicator is color coded in the red, green, and blue color scheme and appears in the lower left corner of a view panel.

Main Menu bar



Tools and items are accessible from pull down menus located at the top of the user interface. In Maya, menus are grouped into menu sets. These menu sets are accessible from the Main Menu bar.
The Main Menu bar appears at the top of the Maya interface directly below the Maya title bar and displays the chosen menu set. Each menu set corresponds to a module within Maya: Animation, Polygons, Surfaces, Rendering, and Dynamics. Modules are a method for grouping related features and tools.
Switching between menu sets is done by choosing the appropriate module from the menu selector on the Status Line (located directly below the File and Edit menus). As you switch between menu sets, the right-hand portion of the menus change, but the left-hand portion remains the same; the left-hand menus are common menus to all menu sets. The left-hand menus contain File, Edit, Modify, Create, Display, and Window. On the Status line, if Animation is selected, the Main Menu changes to display the menu set that relates to the Animation module. In particular, menu titles such as Animate, Deform, Skeleton, Skin, and so on, appear. On the Status line, if Polygon is chosen, the main menu changes to display the menu set for Polygons. Menu titles such as Select, Mesh, Edit Mesh, and so on, appear.

Status Line
The Status Line, located directly below the Main Menu bar, contains a variety of items, most of which are used while modeling or working with objects within Maya. Many of the Status Line items are represented by a graphical icon. The icons save space in the Maya interface and allow for quick access to tools used most often.

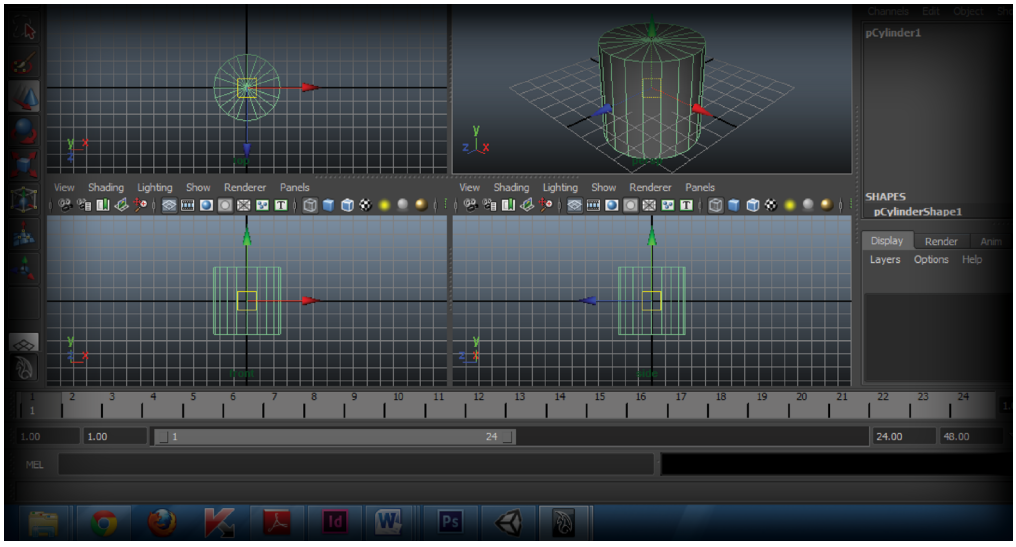
- The first item on the Status line are the Menu Selector used to select between menu sets.
- The third and fourth group of buttons are used to control how you can select objects and components of objects.
- The fifth group of icons are used to control the Snap Mode for objects and components.
- The last section comprise three buttons that are used to show or hide editors, including the Attribute Editor, Channel Box, Layer Editor, and Tool Settings. The default display shows the Channel Box and the Layer Editor. When an object is created, like the cube for example, information about that object displays in these editors.

Shelf
The Shelf is located directly below the Status line. The Maya Shelf is useful for storing tools and items that are used frequently. Maya has some of the Shelf items pre-configured for individual use.

2.3 Other Softwares

2.3.1 Adobe Photoshop CS6

Adobe Photoshop is a graphics editing program developed and published by Adobe Systems.Adobe's 2003 "Creative Suite" rebranding led to Adobe Photoshop 8's renaming to Adobe Photoshop CS. Thus, Adobe Photoshop CS6 is the 13th major release of Adobe Photoshop. The CS rebranding also resulted in Adobe offering numerous software packages containing multiple Adobe programs for a reduced price. Adobe Photoshop is released in two editions: Adobe Photoshop, and Adobe Photoshop Extended, with the Extended having extra 3D image creation, motion graphics editing, and advanced image analysis features.

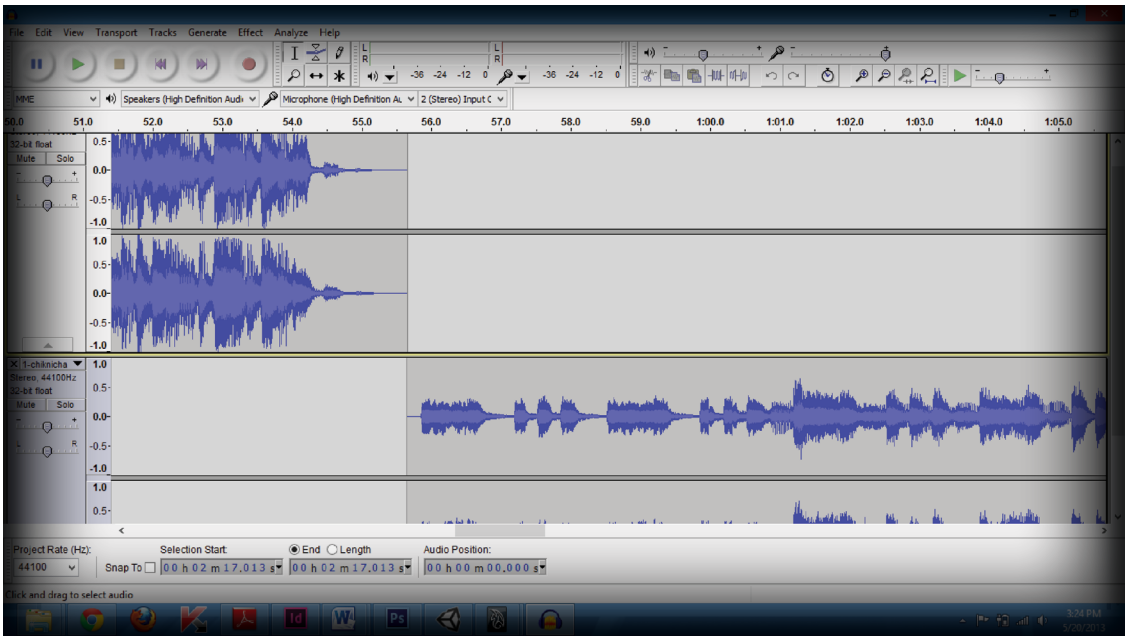


Alongside Photoshop and Photoshop Extended, Adobe also publishes Photoshop Elements and Photoshop Lightroom, collectively called "The Adobe Photoshop Family". In 2008, Adobe released Adobe Photoshop Express, a free web-based image editing tool to edit photos directly on blogs and social networking sites; in 2011 a version was released for the Android operating system and the iOS operating system

2.3.2 Audacity

Audacity is a free, easy-to-use and multilingual audio editor and recorder for Windows, Mac OS X, GNU/Linux and other operating systems.
It has following uses:
•Record live audio.
•Convert tapes and records into digital recordings or CDs.
•Edit Ogg Vorbis, MP3, WAV or AIFF sound files.
•Cut, copy, splice or mix sounds together.
•Change the speed or pitch of a recording.

Audacity is free software, developed by a group of volunteers and distributed under the GNU General Public License (GPL). Free software is not just free of cost rather it is free as in freedom. Free software gives you the freedom to use a program, study how it works, improve it and share it with others.
Programs like Audacity are also called open source software, because their source code is available for anyone to study or use. There are thousands of other free and open source programs, including the Firefox web browser, the OpenOffice.org office suite and entire Linux-based operating systems such as Ubuntu.
Audacity can be used to perform a number of audio editing and recording tasks such as making ringtones, mixing stereo tracks, transferring tapes and records to computer or CD, splitting recordings into separate tracks and more. Vendors can also freely bundle Audacity with their products or sell or distribute copies of Audacity under the GNU General Public License (GPL).



Chapter 3

3D Modelling

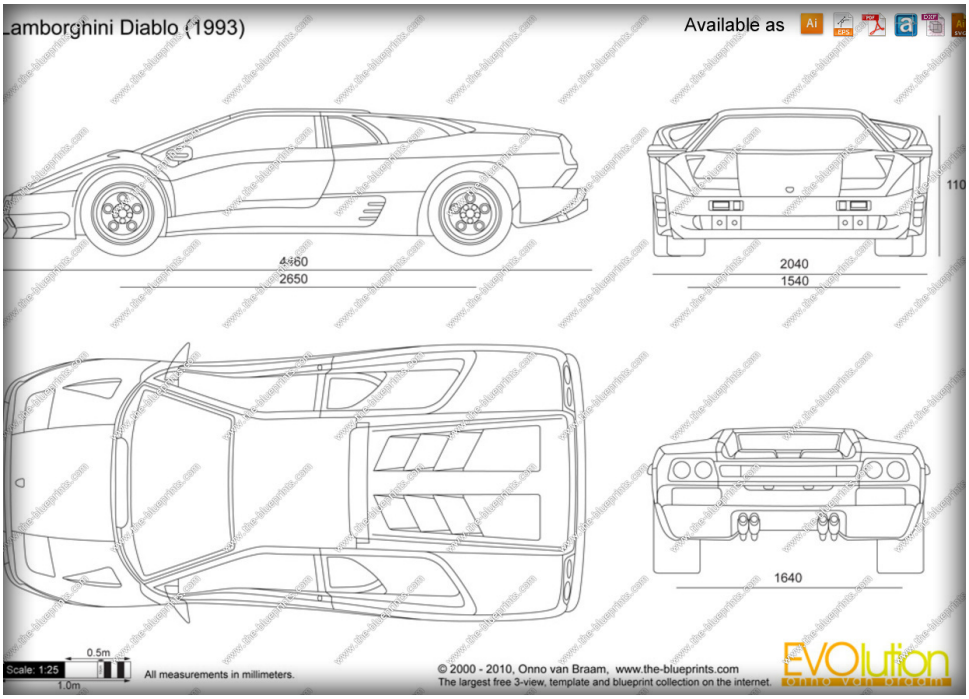
3.1 Modelling the Car

3.1.1 3D Authoring Software

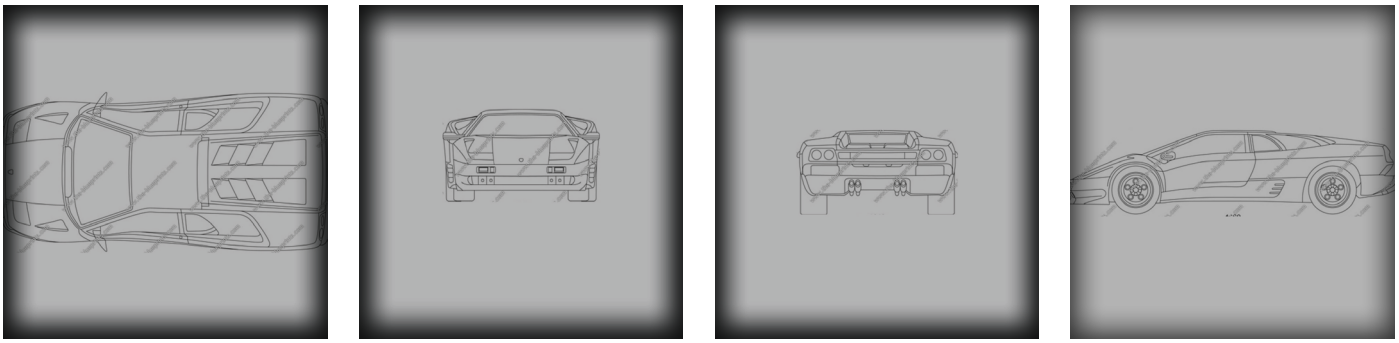
We make use of The Autodesk Maya 2012 3D authoring software. The Choice of software was very clear because of the simplicity of the software for beginners and the wealth of learning material in the form of the largest encyclopedia in the world, the Internet.

3.1.2 Blueprints

To begin designing a car we needed a reference. We cannot start modelling a car unless we have a blueprint of the car indicating the accurate measurements and proportions of the car. A website dedicated to storing blueprints of differnt objects is :- http://www.the-blueprints.com/. Here you can get access to a large database of car blueprints. Paid and Free. Preferably choose the ones with all four views: Top, Front, Rear and side.



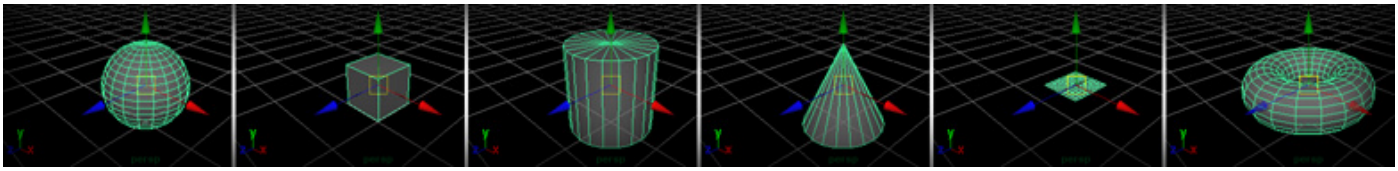
Once we obtained the blueprints, we had to seperate it into the four views. We have to be careful at this stage since we need the blueprints to be proportional to each other. We make use of Adobe Photoshop to Crop and propotionally scale each view into a 1024x1024 image.



Once we get the 4 images, we can load each image as an image plane in Maya corresponding to the 3 views, The Front and the rear view is Shared. If done correctly, then the base of the entire 3D Model construction is complete. We then proceed to build the model.

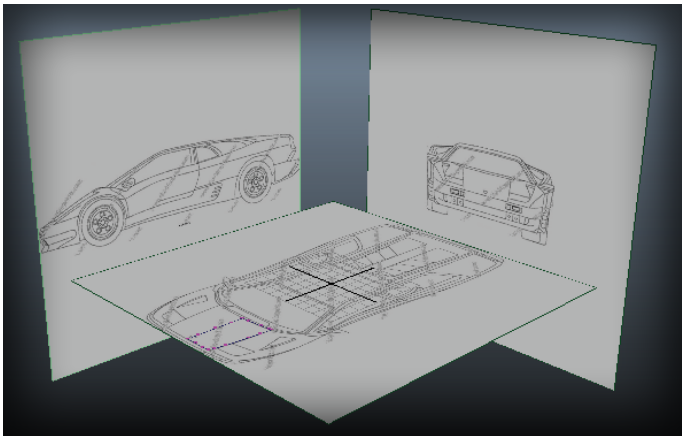
3.1.3 Polygon Primitives

Maya includes 12 polygon primitives. Six of these are similar to their NURBS or subdiv equivalents: sphere, cube, cylinder, cone, plane, and torus. The other six are more complex shapes, which are available only in polygonal form: prism, pyramid, pipe, helix, soccer ball, and platonic solids.



3.1.4 Image Planes

The four images coresponding to the three views are setup so that vertices can be acurately traced with respect to the three images.

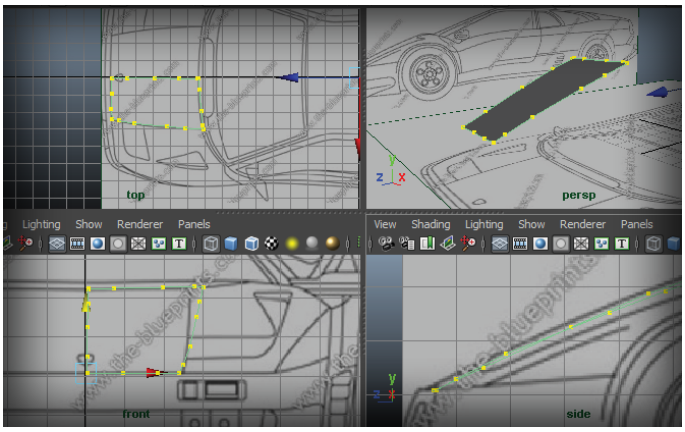


3.1.5 Create Polygon Tool

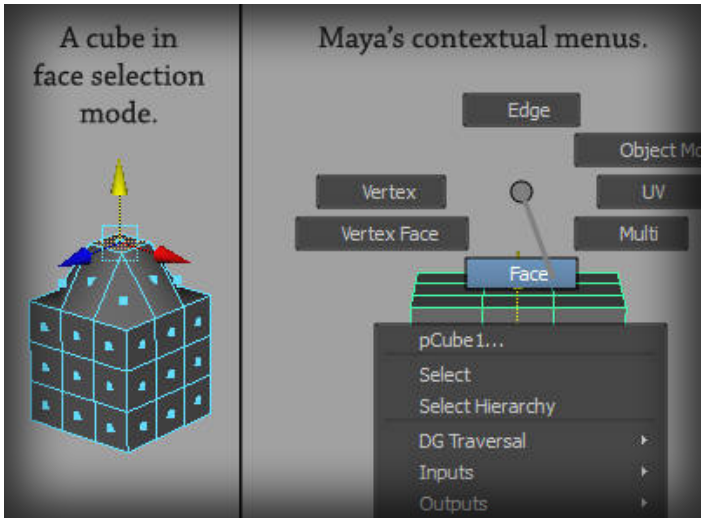
Lets you create individual polygons by placing vertices in the scene view. We create polygons by tracing points along the image planes. The result is a flat plane with numerous verices on the boundary that could be manipulated in 3D space.

3.1.6 Transformations

There are three basic forms of object manipulation in any 3D application—translate (or move), scale, and rotate. Each of these tools, including the Universal Manipulator tool, has axes that you can grab and move to transform an object. These axes, called manipulators, let you translate (move), rotate, or scale the object. Manipulators make it easy to constrain objects along a particular axis: You click and drag the colored line for the axis along which you want to constrain the object. The colors remain consistent for each tool. RGB colors coincide with the x-y-z axes: The manipulator's x axis is red, the y axis is green, and the z axis is blue. If you forget an axis's color, check the View axis in the lower left-hand corner of each pane or the View Compass at upper right. The axis selected on the manipulator is always yellow.



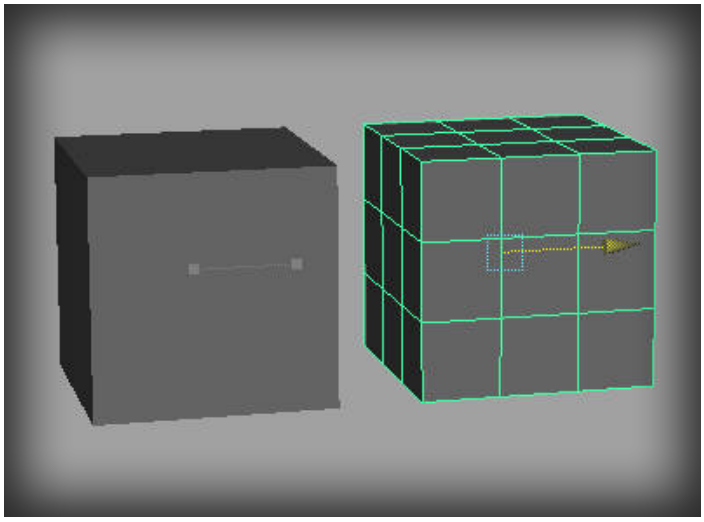
3.1.7 Selection Modes



Maya has a number of additional selection types, and each one is used for a different set of operations. To access Maya's other selection modes, hover your mouse pointer over the cube and then click and hold the right mouse button (RMB). A menu set will appear, revealing the Maya's component selection modes—Face, Edge, and Vertex being the most important.

In the fly menu, move your mouse to the Face option and release the RMB to enter face selection mode. You can select any face by clicking its center point and can then use the manipulator tools to modify the shape of the model. Select a face and practice moving, scaling, or rotating it like we've done in the example above. These same techniques can also be used in edge and vertex selection mode. Pushing and pulling faces, edges, and vertices is probably the single most common function you'll perform in the modeling process

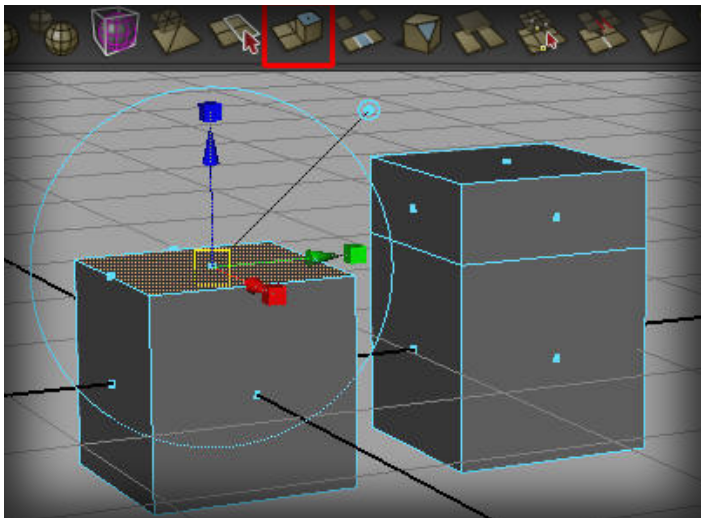
3.1.8 Duplication



Duplicating objects is an operation you'll use over, and over, and over throughout the modeling process. To duplicate a mesh, select the object and press Ctrl + D. This is the simplest form of duplication in Maya, and makes a single copy of the object directly on top of the original model.

3.1.8 Extrusion

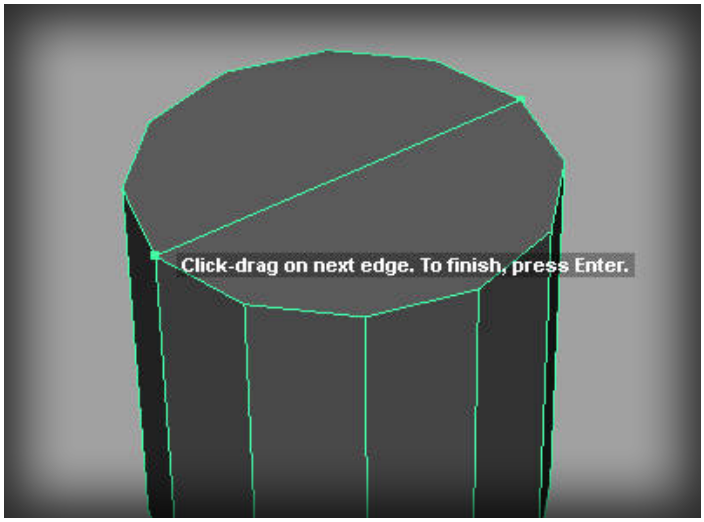
Extrusion is our primary means of adding additional geometry to a mesh in Maya.The extrude tool can be used on either faces or edges, and can be accessed at Mesh Extrude, or by pressing the extrude icon in the polygon shelf at the top of the viewport



Switch into face mode, select the upper face, and then press the extrude button in the polygon shelf.A manipulator will appear, which looks like an amalgamation of the translate, scale, and rotate tools. In a sense it is—after performing an extrusion, it is essential that you either move, scale, or rotate the new face so that you don't end up with overlapping geometry.

Notice that there's no global scale manipulator at the center of the tool. This is because the translate tool is active by default.

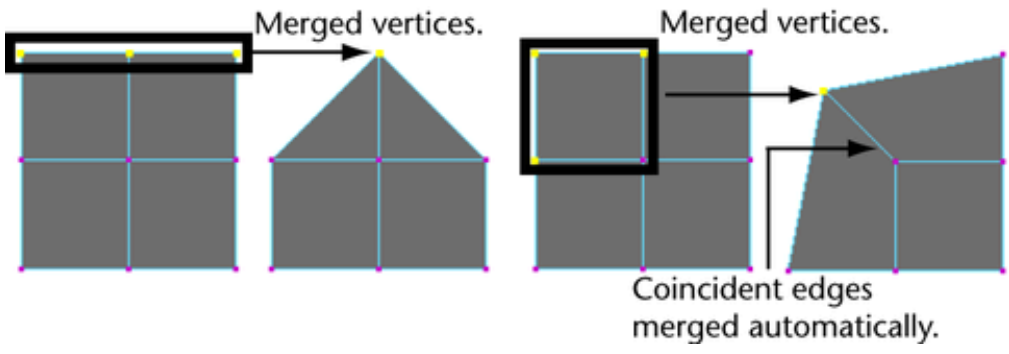
3.1.9 Split Polygon Tool



With the cylinder in object mode, go to Edit Mesh Split Polygon Tool. Our goal is to break down the 12-sided face into four-sided quads by creating new edges between existing vertices. To create a new edge, click on a border edge and (still holding down the left mouse button) drag the mouse toward the starting vertex. The cursor should lock onto the vert.
Perform the same action on the vertex directly across from the first and a new edge will appear, dividing the face into two halves.
To finalize the edge, hit Enter on the keyboard.
Note: An edge is never finalized until you strike the enter key. If you'd clicked on a third (or fourth, fifth, sixth, etc.) vertex without first clicking enter, the result would have been a series of edges connecting the entire sequence of vertices. In this example we want to add the edges one-by-one.

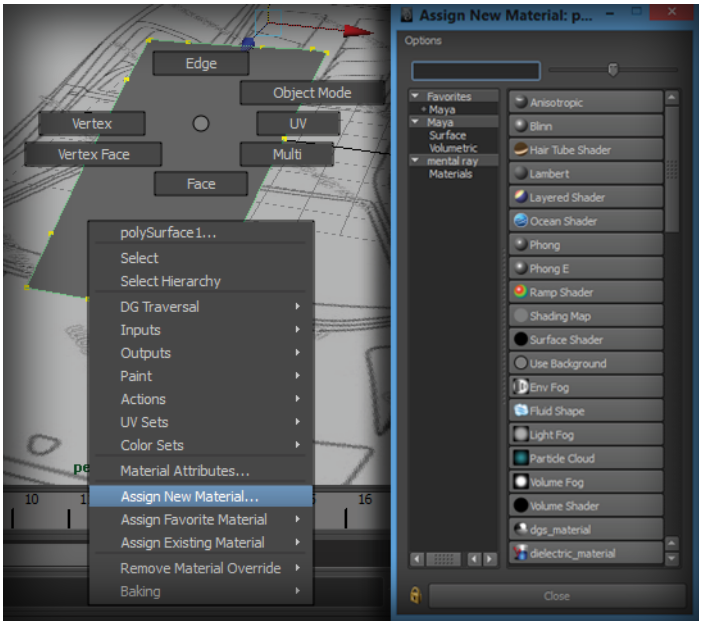
3.1.10 Merge Vertex Tool

You can merge two vertices on the same mesh together using the Merge Vertex Tool. Merging two vertices with this tool yields the same result as selecting the vertices individually and using the Merge feature.
To merge two vertices using the Merge Vertex Tool
Select a mesh.
Set the component selection mask to vertices.
Select Edit Mesh > Merge Vertex Tool.
Drag the mouse from a source vertex on the selected mesh to a destination vertex.
The vertices merge to become a single vertex.
When dragging between the vertices you want to merge, a red line appears from the source vertex to the mouse cursor.
You can also merge multiple vertices with the Merge Vertex Tool.
To merge multiple vertices
Select a set of vertices with the Select Tool.
Select Edit Mesh > Merge Vertex Tool.
Drag the the mouse from one of the source vertices to a destination vertex.
All the selected source vertices merge with the destination vertex.
You can change the behavior of the Merge Vertex Tool, with the Tool Settings Editor, to merge the vertices at either the destination vertex or at the center between the source and destination vertices. In the latter case, a small red dot will appear at the center of the red line drawn between the source vertex and the mouse cursor.



3.1.11 Materials

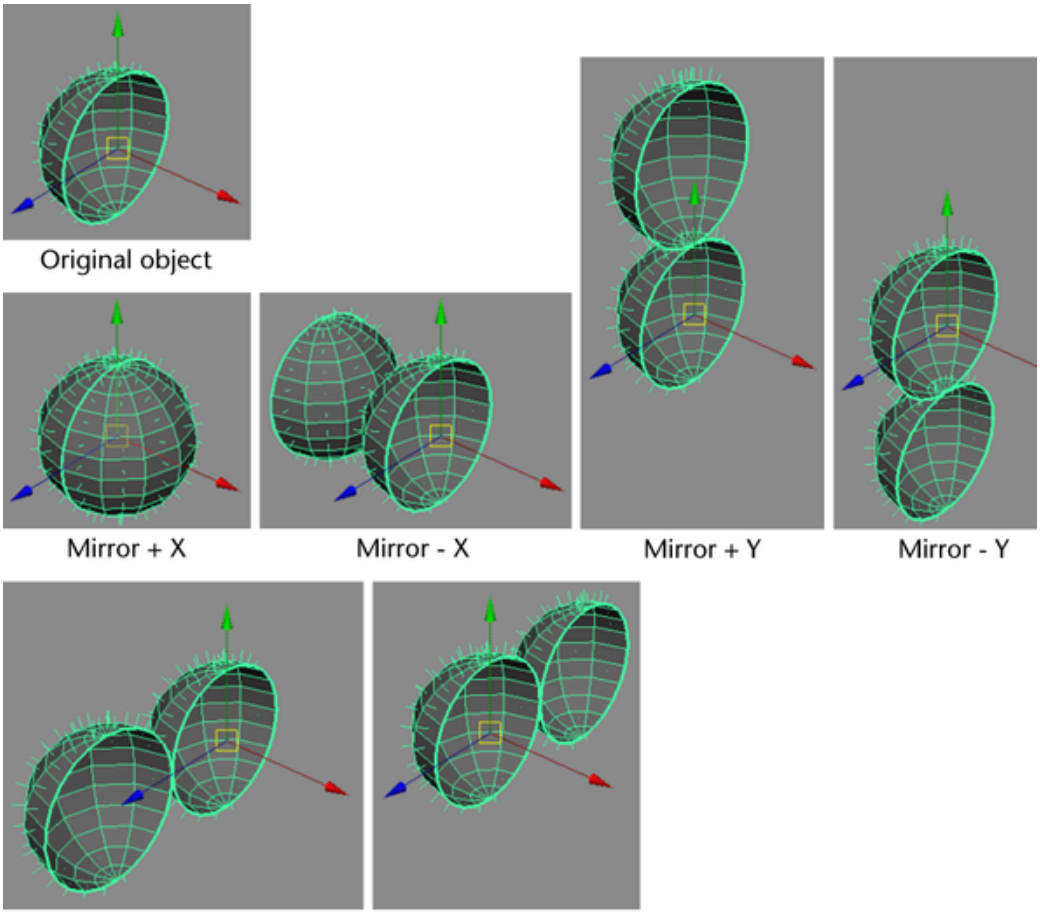
Though 3D surfaces in Maya respond to light similarly to those in the real world, there are important differences in the way that surfaces and lights interact in computer graphics software. Material nodes are a type of render node (see Render nodes) that, when applied to an object, let you define how the object's surface appears when rendered.
In Maya, material nodes define how surfaces react to light. Maya contains several types of material nodes that help you simulate the real-world qualities or behaviors of surfaces to light: Surface material nodes, Displacement material nodes, and Volumetric material (atmosphere) nodes.



Surface material
Surface materials represent the types of surfaces onto which you can map textures. Attributes such as shininess, matte, reflectivity, glossiness, and so on, vary among the different types of materials in Maya.
For example, if the texture requires a shiny surface, such as chrome, use a Phong material rather than something like a Lambert.
Right-Clickthe object and select
Applying Materials
Assign New Material. The Assign New Material window appears. Choose from the available materials.
Alternatively Right-Click the object and select Assign Favorite Material.
Choose among the materials from your Favorites list.

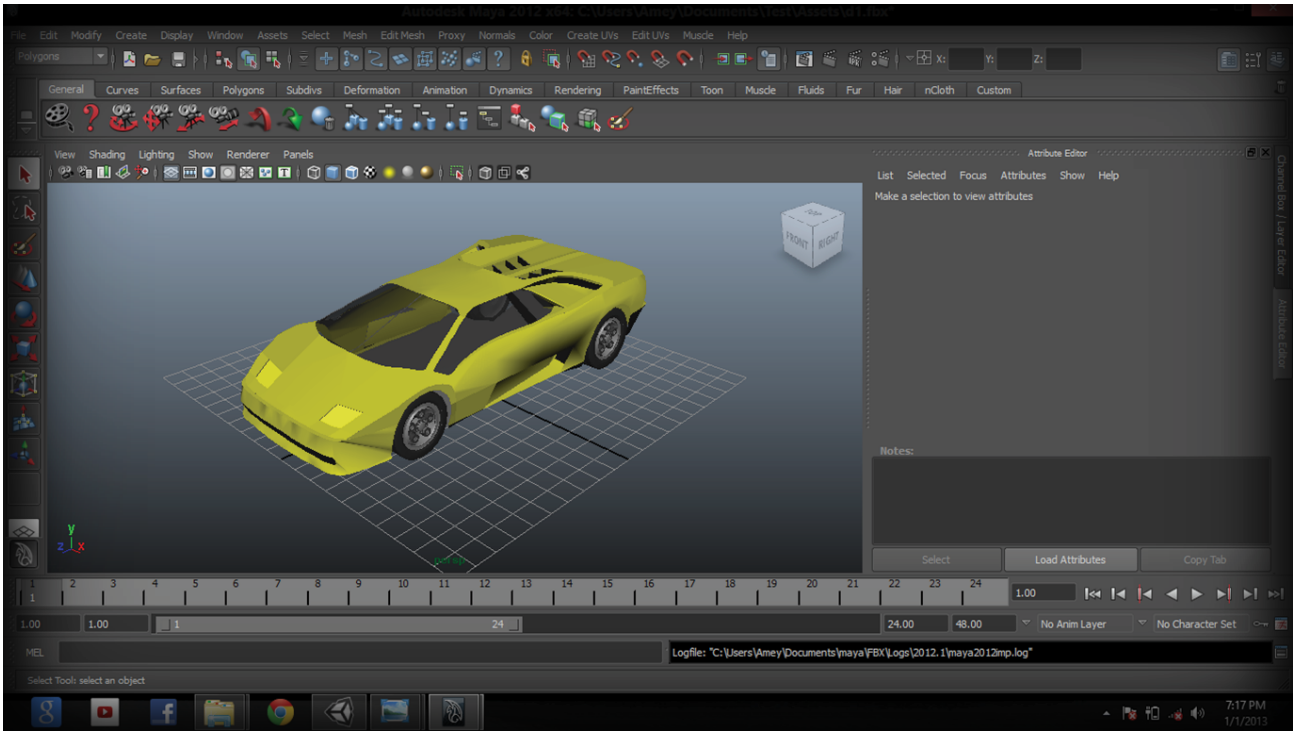
3.1.12 Mirror Geometry Tool

Mirror direction
Specifies the direction you want Maya to mirror the selected polygonal object. By default, the direction is +X. Change these options and click Mirror if you want to mirror the object in another direction.
Merge with the original
Selects how you want the polygons to merge.
When turned on (the default setting), Maya duplicates and flips the original polygon and merges the duplicated polygon with the original polygon. This makes the new polygonal object one shell. When turned off, Maya duplicates and flips the original polygon but does not merge the separate shells.
Merge vertices
Select this option to merge adjacent vertices, creating a single shell.
Connect border edges
Select this option to connect the original polygon with the mirrored polygon at the border edges, filling in faces to create a closed shape.



3.1.13 Wrapping Up

By applying the above mentioned tools, we are able to sculpt the entire model of the vehicle. We make sure that the entire model is divided into 5 distinct objects ie. The Car body and the Four wheels. Also we have to make sure that we delete history every now and then since it takes quite a bit of memory. Also Apply Center Pivot to the objects, to make sure their origins are in their centers.

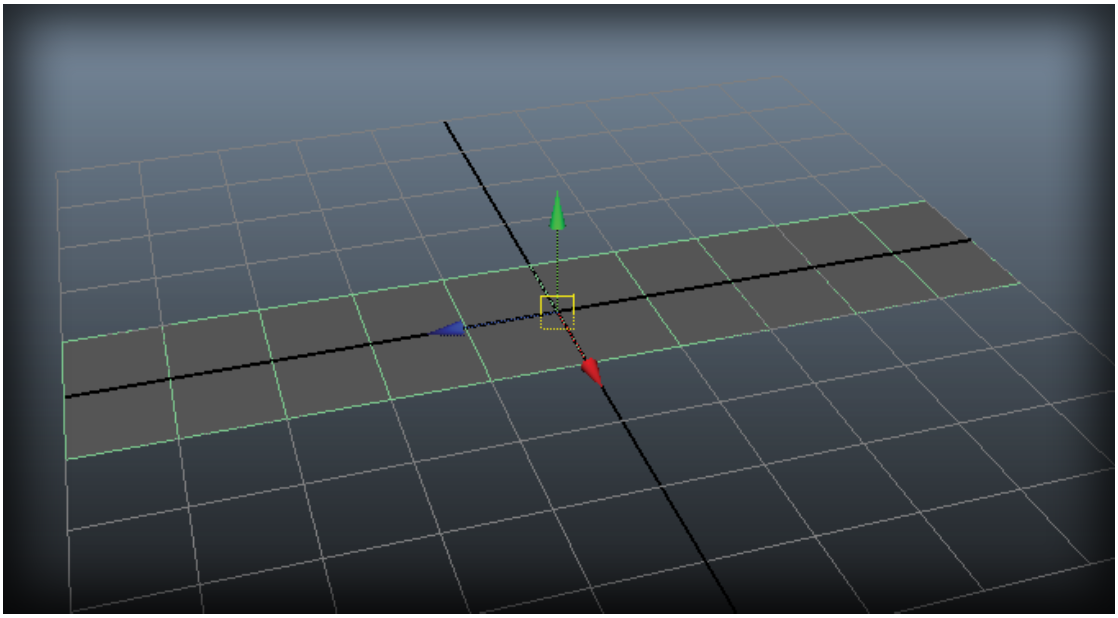


3.2 Modelling the Track

The approach to modelling the track is slightly different to modelling the car. Here we do not use blueprints, or any kind of other reference material. The final model unlike the car consists of the two parts.

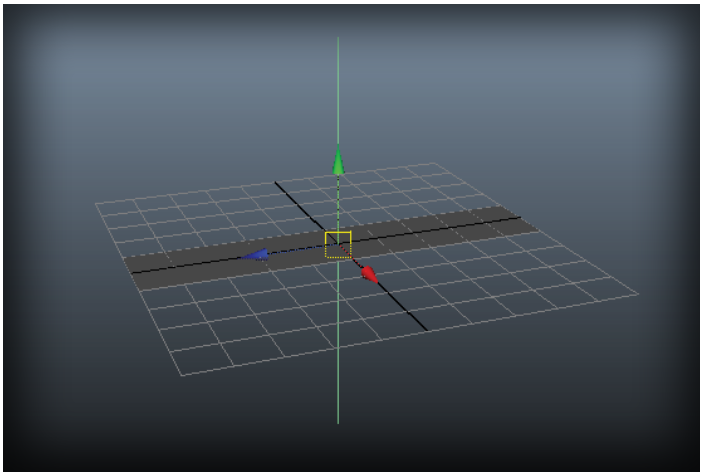
3.2.1 Creating a Plane

Choose the type of plane you want to construct on the Maya grid. There are two types of planes: the surface plane and polygon plane. Select the "Polygon" tab from the main tab menu in Autodesk Maya. Click on the "Polygon Plane" option in the polygon tab menu. It is the flat tablecloth-looking icon. If you are confused on which icon it is, simply move the mouse over the icon to view the description. Move the mouse over the grind and hold down the left mouse button to draw the plane. Once you have the correct dimensions for the plane, release the left mouse button.

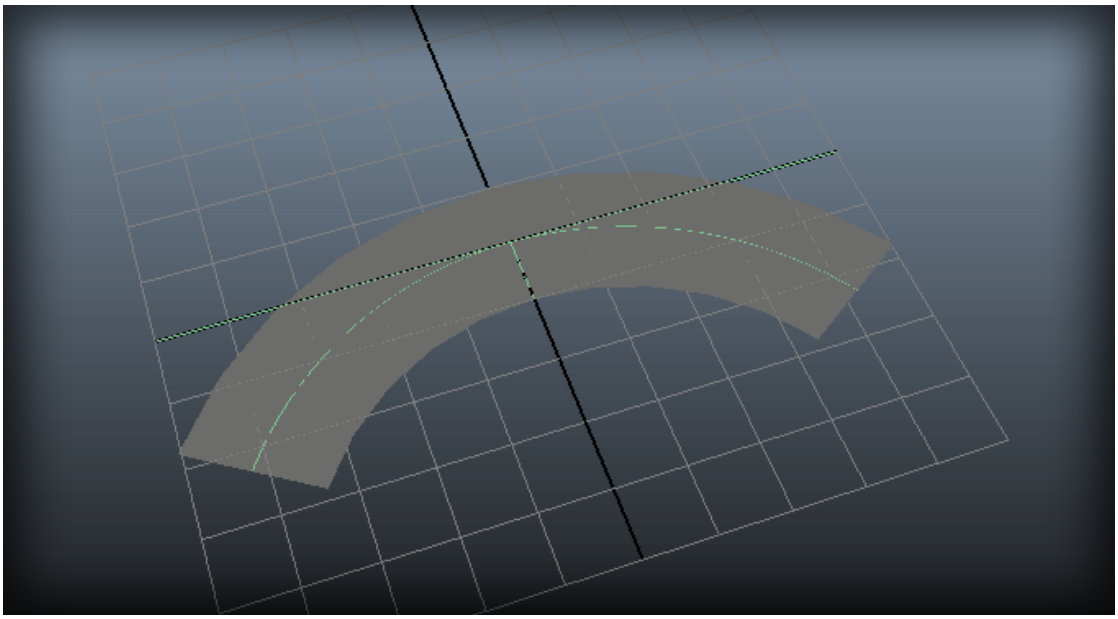


3.2.2 Creating Bends

To create a bend deformer. Select the object(s) you want to deform. Select Create Deformers > Nonlinear > Bend > . The Create Bend Deformer Options window appears. Click the Basic and Advanced tabs and set the creation options.



To create bend deformation effects
Manipulate the bend deformer handle.
Edit bend deformer channels and attributes.
For more information on creating and editing deformation effects, see Edit bend nonlinear deformers.



3.2.3 Combining and Merging

It is often handy to merge two separate polygon objects into a single polygon mesh.

Step One

Move polygon objects close to each other. Shift-click to select both objects and combine them into one object (Mesh > Combine) from the Polygons menu set.

Step Two

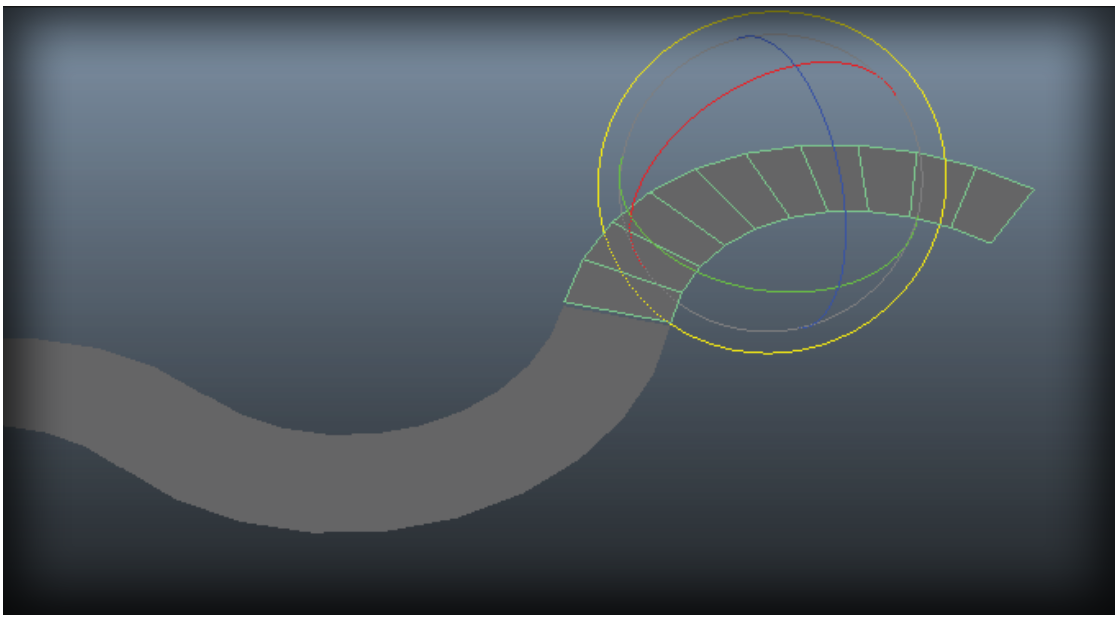
With the object selected, click the Component, Vertices icon.

Step Three

Align the points and edges that will be joined together in Component Mode. Use the Snap to Point feature (v key) to snap to point while moving points.

Step Four

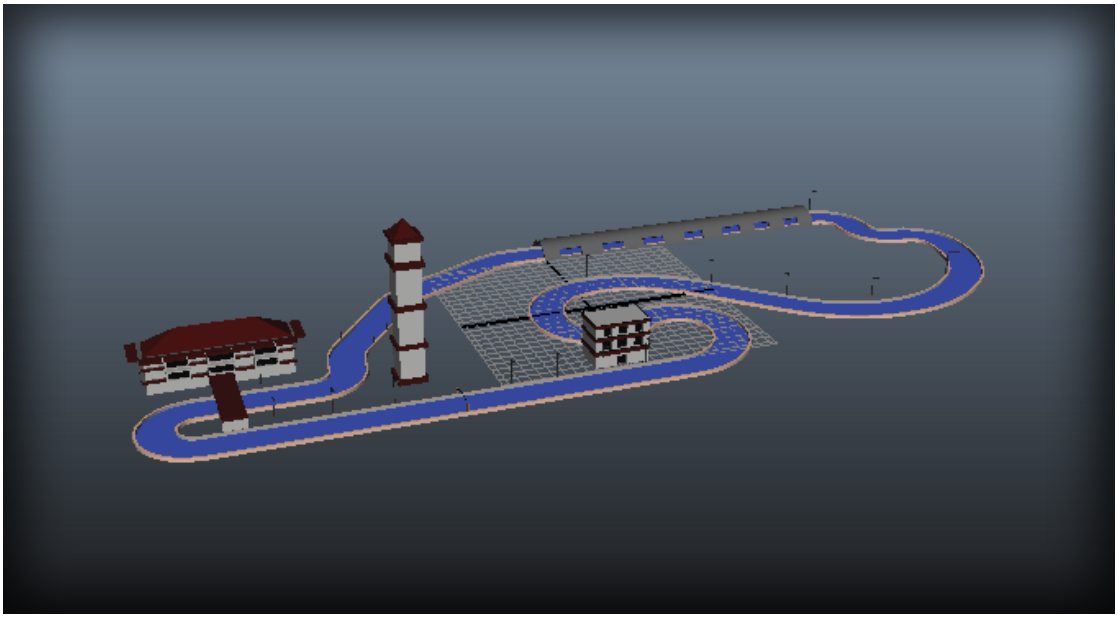
Return to the Object mode, select the object and choose Edit Mesh > Merge from the Polygons menu set. Set Threshold Distance in options if necessary. The Threshold Distance sets the maximum distance that two vertices can be apart to be merged.



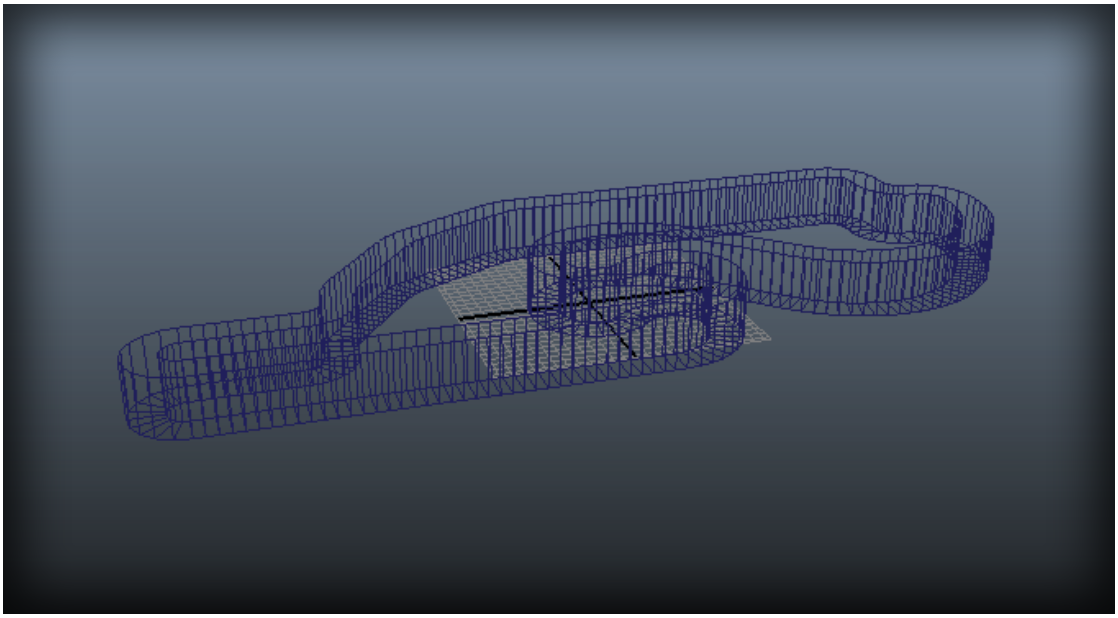
3.2.4 Artwork & Collider Track

Once all the planes are combined and their vertices merged, the final thing to do is to seperate it out into two different tracks:- 1) Artwork track 2) Collider track

The Artwork track is just the visual render of the track. This track will contain all the visual aspects of the track. Apart from just the track, it also includes the buildings, streetlamps, and any other additional models that only contributes to the visual atmosphere rather than interaction with the car model.



The Collider track on the other hand is used as a mesh in unity for the actual interaction with the car model. It however does not contain a renderer and hence wont be visible.
The two tracks are combined in unity to form a seamless custom collision system that restricts the car within the track.

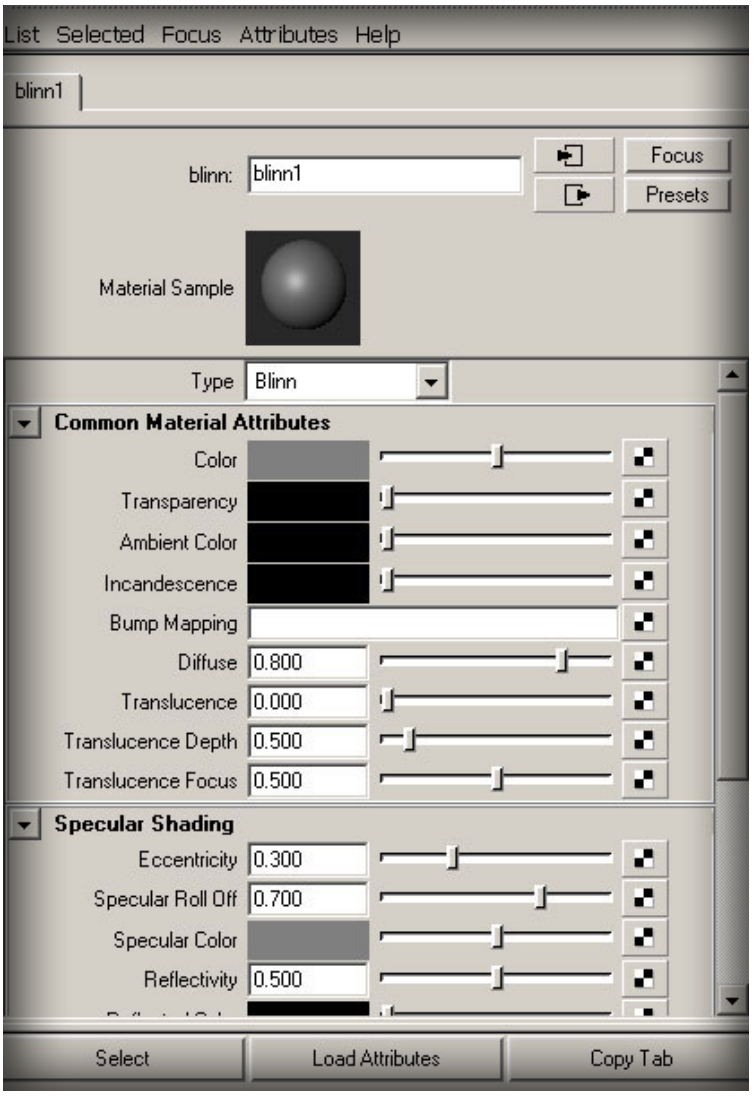


3.3 Materials

3.3.1 Basic Material Types

Lambert
Lambert is a flat material type that yields a smooth look without specular highlights. It calculates without taking into account surface or reflectivity, which gives a matte, chalk-like appearance. Lambert material is ideal for surfaces that don't have highlights: pottery, chalk, matte paint, and so forth. By default, any newly created object is assigned the Lambert shader.

Phong
The Phong material takes into account specular reflectivity to create highlights across an object surface. The algorithm can be customized for surfaces such as plastic, porcelain, and glazed ceramic.



PhongE
PhongE is a faster rendering version of Phong that yields somewhat softer highlights than Phong.

Blinn
The Blinn material calculates highlights on surfaces similarly to Phong; however, Blinn can achieve a more accurate representation of the soft tinted highlights you see on metallic surfaces. Because Blinn is a versatile material type and doesn't cause flickering with bump maps.

Anisotropic
The Anisotropic material type stretches highlights and rotates them based on the viewer's position. Anisotropic materials are ideal for materials such as hair, feathers, brushed metal, and satin.

Ramp Shader
The Ramp Shader material consists of built-in ramp graphs to offer more advanced control and simplify the shader network.

Ocean Shader
The Ocean Shader material has several items in the shader's attributes that control how the material behaves over time, and it has graphs to add detail to the base shader. Attributes include Wave Speed, Wave Height, Wave Turbulence, and Wave Peaking.

Layered Shader
The Layered Shader lets you combine several materials to create a more complex material.

Shading Map
The Shading Map material is primarily designed to let you get a "cel" look in 3D, like typical animated cartoons.

Surface Shader
The Surface Shader is used when you want to control a material's color, transparency, and glow with something else in Maya.

Use Background
The Use Background shader cuts a "hole" in the image's alpha channel where objects with the material appear. This material is useful for combining separately rendered images in a compositing program to create the final results.

3.3.2 Material Settings

To edit material settings, double-click on any material in Hypershade's top or bottom tabs. Usually, you create a Blinn material in the Work Area of the bottom tab panel, and then double-click it to edit it in the Attribute Editor.
Notice the material name at the top of the Attribute Editor, which Maya sets to blinn1 for a default starting name. Maya increments the number if you create more Blinn materials. Next is the Common Material Attributes section, followed by the Specular Shading section. These two sections, displayed by default, are used the most in material editing.
Color
The default material color.
Transparency
If the transparency value is 0(black),the surface is totally opaque; if the transparency value is 1(white),the surface is totally transparent. If you change transparency from the default black ,the background of the material's hypershade swatch becomes a checked pattern .This is not a visual aid and is not rendered.
Ambient Color
Uses Black by default. As the ambient color becomes lighter, it affect's the material's color by lightening it and blending the two colors.
Incandescence
The color and the brightness of light that a material appears to be emitting. The default color value is 0.
Bump Mapping
Makes the surface appear more rough or bumpy by altering surface normals according to the intensity of the pixels in the bump map texture. A bump map does not actually alter the surface.
Diffuse
Gives the material ability to reflect light in all directions. The diffuse value acts like a scaling factor applied to the color setting-the higher the diffuse value, the closer the actual surface is to the color setting.
Translucence
Gives the material the ability to transmit and diffuse light. Light falling on a translucent surface is first absorbed beneath the surface and then diffused in all directions.

3.3.3 Surface Material Attributes

Specular color
The color of shiny highlights on the surface. A black specular color produces no surface highlights. The default color value is 0.5.

Reflectivity
Gives the surface the ability to reflect its surroundings or the reflected color. The default color value is 0.5.

Reflected color
Represents the color of light reflected from the material. This can be used to tint a reflection.

Anisotropic
Represents surfaces with grooves, such as cd or feathers etc. Anisotropic material (such as Phong or Blinn)reflects specular light identically in all directions.If you spin an isotropic sphere, its specular highlight remains still.

Roughness
Determines the overall roughness of the surface. The range is 0.01 to 1.0.The default is 0.7.Smaller values correspond to smoother surfaces and the specular highlights are more concentrated.

Fresnel Index
A fresnel is a flat lens consisting of a number of concentric rings that reduces spherical abnormalities. The Fresnel index for water is 1.33.Value range from 1.0 to 20.0.

Anisotropic Reflectivity
If on, Maya automatically calculates Reflectivity as a fraction of roughness. Reflectivity is on by default.

Chapter 4

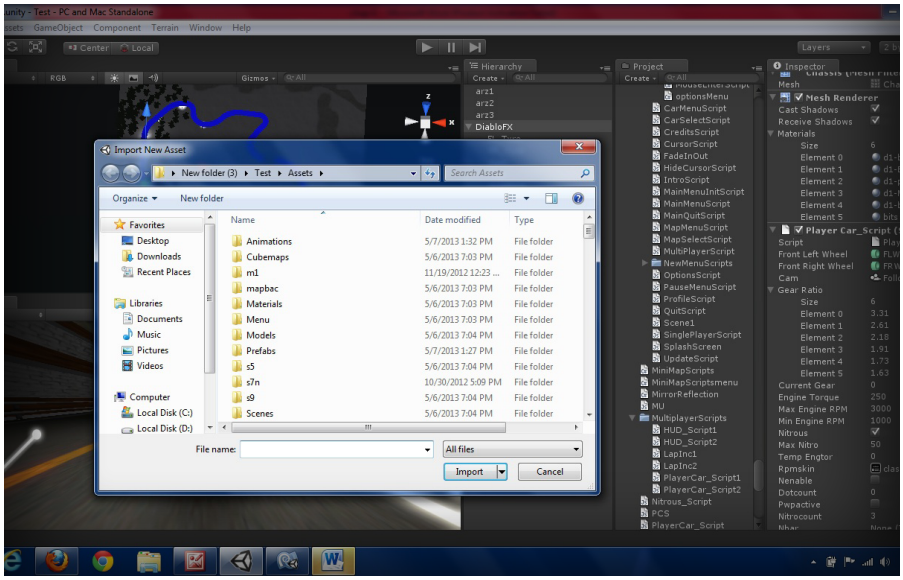
Scene Setup

4.1 Importing Assets

Unity will automatically detect files as they are added to your Project folder's Assets folder. When you put any asset into your Assets folder, you will see the asset appear in your Project View.

Asset Types

There are a handful of basic asset types that will go into your game. The types are:
Meshes & Animations : Whichever 3D package you are using, Unity will import the meshes and animations from each file.
Textures : Unity supports all image formats. Once your texture has been imported, you should assign it to a Material.
Sounds : Unity features support for two types of audio: Uncompressed Audio or Ogg Vorbis. Any type of audio file you import into your project will be converted to one of these formats.



File Type Conversion

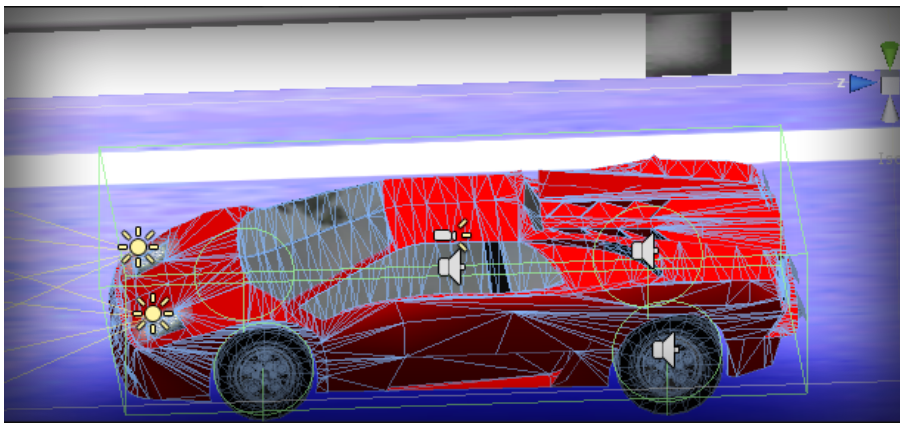
.AIFF	Converted to uncompressed audio on import, best for short sound effects.
.WAV	Converted to uncompressed audio on import, best for short sound effects.
.MP3	Converted to Ogg Vorbis on import, best for longer music tracks.
.OGG	Compressed audio format, best for longer music tracks.

In the project, the car model(asset) has been placed in the asset folder of the project with the .mb extension. If the car model is detected as a valid file-type, then unity will launch Maya in the background. Unity then communicates with Maya to convert the .mb file into an FBX format which Unity can read. The first time you import a Maya file in Unity, Maya has to launch in a command line process, this can take around 20 seconds, but subsequent imports will be very quick.

4.2 Materials and Shaders

When a 3D model is imported, Unity represents it as many different objects, including a hierarchy of GameObjects. A Mesh must be attached to a GameObject using a Mesh Filter component. For the mesh to be visible, the GameObject must also have a Mesh Renderer or other suitable rendering component attached. With these components in place, the mesh will be visible at the GameObject's position with its exact appearance dependent on the Material used by the renderer. The Mesh Renderer takes the geometry from the Mesh Filter and renders it at the position defined by the object's Transform component.

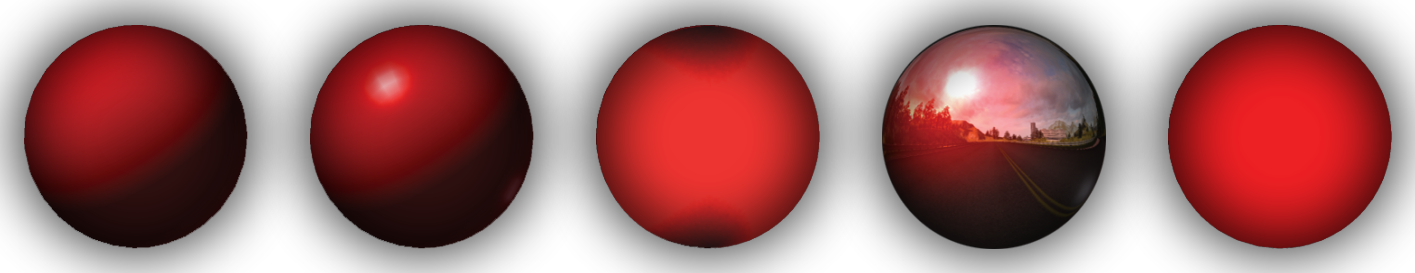
4.2.1 Material



The car model from Maya is imported into unity. Mesh renderer and necessary materials are applied to it. The car model has elements, each for body, tail lights, tires etc. a necessary material is applied to it. Mesh renderer shows the different materials applied to the car model. To create a new Material, use Assets->Create->Material from the main menu or the Project View context menu. Once the Material has been created, you can apply it to an object and tweak all of its properties in the Inspector. You can select which Shader you want any particular Material to use. Simply expand the Shader drop-down in the Inspector, and choose your new Shader.

4.2.2 Shaders

Shaders are small scripts that let you configure the how the graphics hardware is set up for rendering. Unity ships with over eighty built-in shaders. You can extend this set by making your own shaders. Shaders in Unity can be written in one of three different ways:
Surface Shaders : Surface Shaders are your best option if your shader needs to be affected by lights and shadows.
Vertex and Fragment Shaders : Vertex and Fragment Shaders will be required, if your shader doesn't need to interact with lighting, or if you need some very exotic effects that the surface shaders can't handle.
Fixed Function Shaders : Need to be written for old hardware that doesn't support programmable shaders.



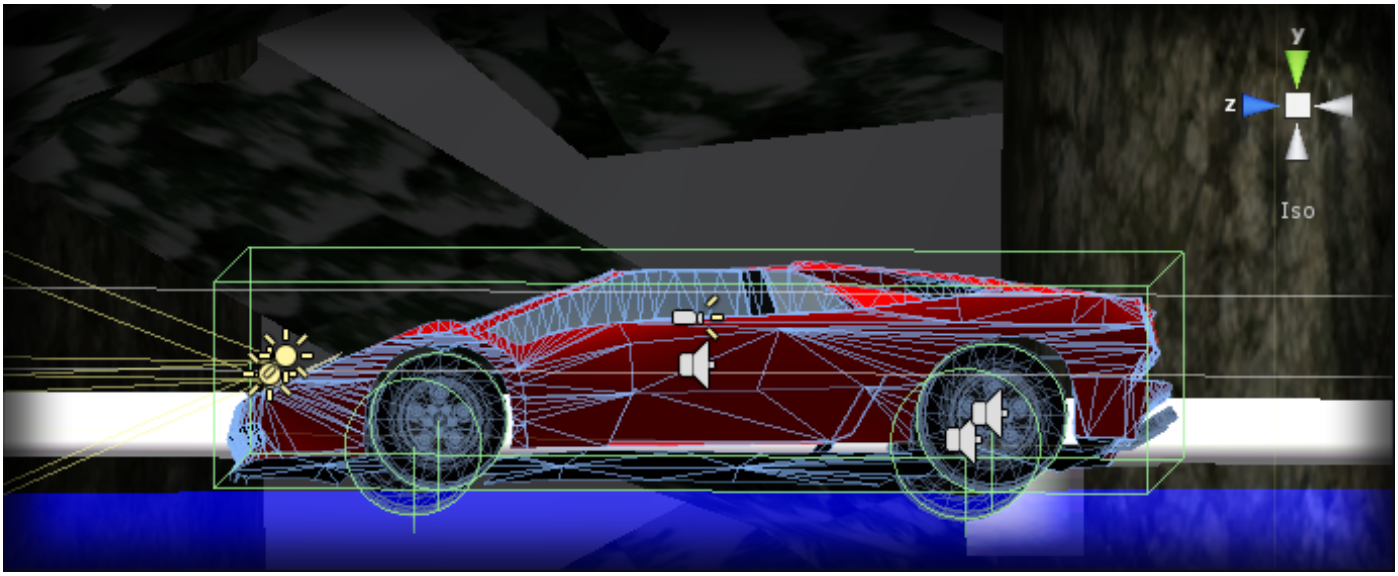
Diffuse | Specular | Particles / Additive | Reflective with Cubemap | Self-Illumin

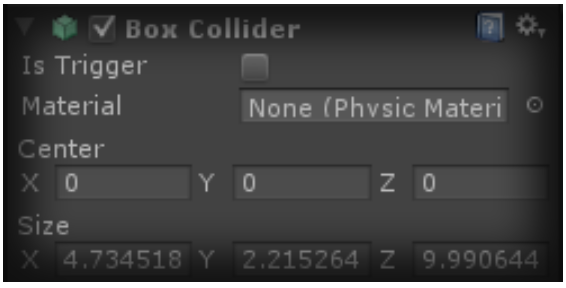
4.3 Setting up the Car Rig

4.3.1 Adding the Box Collider

The Box Collider is a basic cube-shaped collision primitive.

Properties
Is Trigger - If enabled, this Collider is used for triggering events, and is ignored by the physics engine.
Material - Reference to the Physics Material that determines how this Collider interacts with others.
Center - The position of the Collider in the object's local space.
Size - The size of the Collider in the X, Y, Z directions.





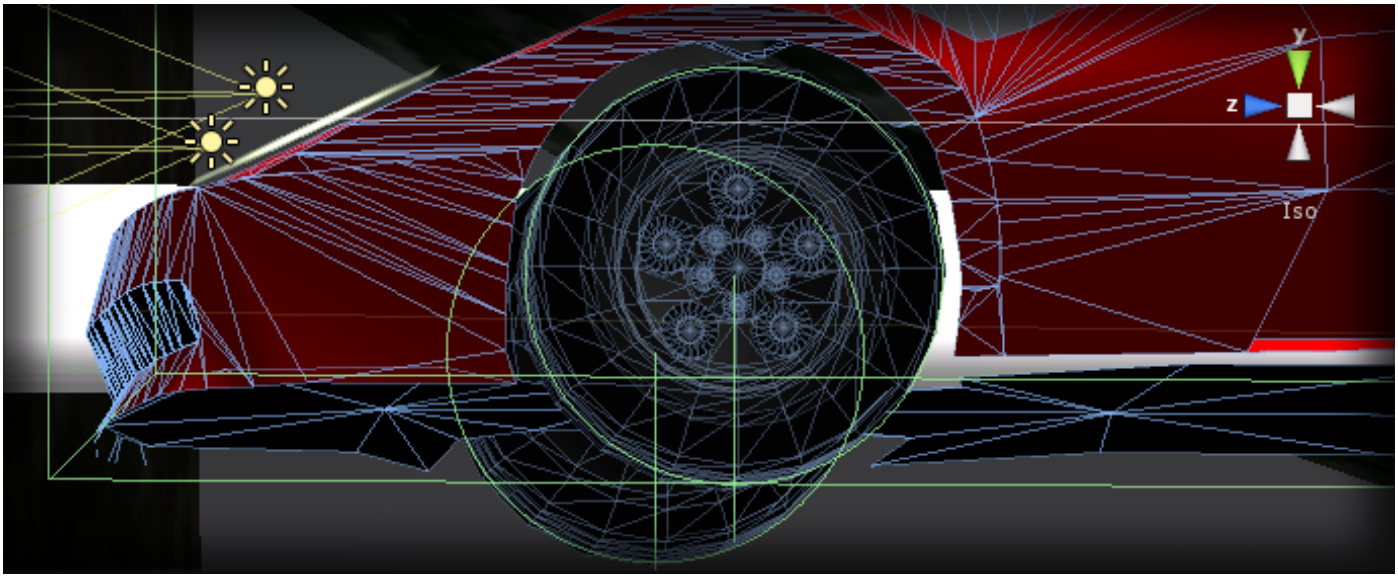
The box collider for the car model is created by selecting the car and Component->Physics->Box Collider. Its properties are then customized depending upon the requirement.

The Box Collider can be resized into different shapes of rectangular prisms. It works great for doors, walls, platforms, etc. It is also effective as a human torso in a ragdoll or as a car hull in a vehicle. Of course, it works perfectly for just boxes and crates as well.

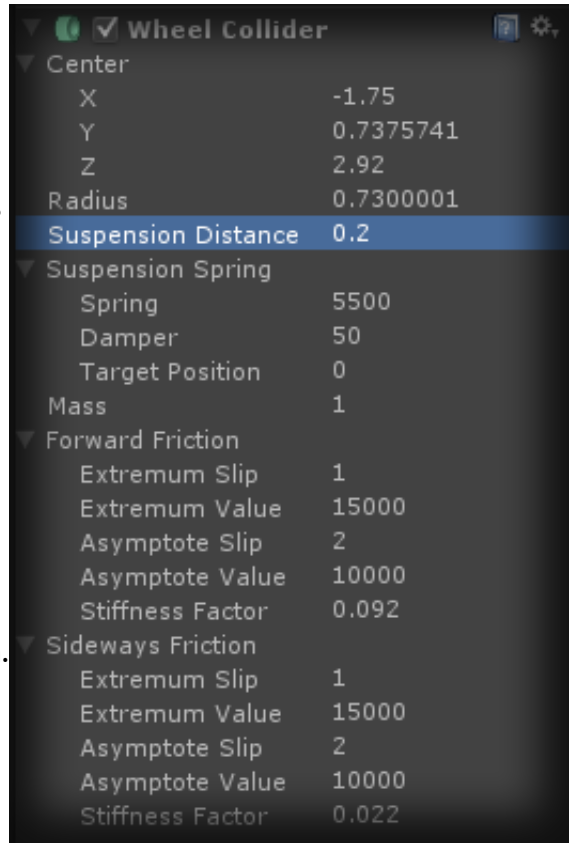
Colliders work with Rigidbodies to bring physics in Unity to life. Whereas Rigidbodies allow objects to be controlled by physics, Colliders allow objects to collide with each other. Colliders must be added to objects independently of Rigidbodies. A Collider does not necessarily need a Rigidbody attached, but a Rigidbody must be attached in order for the object to move as a result of collisions.

4.3.2 Adding the Wheel Collider

The Wheel Collider is a special collider for grounded vehicles. It has built-in collision detection, wheel physics, and a slip-based tire friction model. It can be used for objects other than wheels, but it is specifically designed for vehicles with wheels.



Properties



Center - Center of the wheel in object local space.
Radius - Radius of the wheel.
Suspension Distance - Maximum extension distance of wheel suspension, measured in local space. Suspension always extends downwards through the local Y-axis.
Suspension Spring - The suspension attempts to reach a Target Position by adding spring and damping forces.
Spring Spring force attempts to reach the Target Position. A larger value makes the suspension reach the Target Position faster.
Damper - Dampens the suspension velocity. A larger value makes the Suspension Spring move slower.
Target Position - The suspension's rest distance along Suspension Distance. 0 maps to fully extended suspension, and 1 maps to fully compressed suspension. Default value is zero, which matches the behavior of a regular car's suspension.
Mass - The Mass of the wheel.
Forward/Sideways Friction - Properties of tire friction when the wheel is rolling forward and sideways. See Wheel Friction Curves section below.

The wheel's collision detection is performed by casting a ray from Center downwards through the local Y-axis. The wheel has a Radius and can extend downwards according to the Suspension Distance. The vehicle is controlled from scripting using different properties: motorTorque, brakeTorque and steerAngle.
The Wheel Collider computes friction separately from the rest of physics engine, using a slip-based friction model. This allows for more realistic behavior.

4.3.3 Applying Scripts

To give some life to the models, we need to apply certain scripts to these objects. Scripting inside Unity consists of attaching custom script objects called behaviours to game objects. Different functions inside the script objects are called on certain events. There are numerous scripts applied to the car, The most important of these are mentioned below:-

Player Car Script

function Start ()
- Set the center of mass to make the car more stable.
- Initialize engine torque to 1.

end Start

function Update ()
- Compute the engine RPM based on the average RPM of the two wheels.
- Call the function to shift gears.
- Set the audio pitch to the percentage of RPM to the maximum RPM plus one
  Condition to ensure that the pitch does not reach a value higher than is desired.
- if Nitrous Key is pressed and Nitrous is Available
  then Increase engine torque to 1.7 times.
- if Reset Key is pressed
  then call Resetcar() function.
- The value of engine torque is multiplied by the current gear and by the user input ,and is applied to the wheels of the car.
- Steer angle of the car = Velocity of the car multiplied by Horizontal axis user input.

end Update.

function ShiftGears()
//During Acceleration
- if Engine RPM > Max Engine RPM
  then for all the gears in ascending order
  - if Wheel RPM * gear< Max Engine RPM
  - Increment Gears

//During Deceleration
- if Engine RPM <= Min Engine RPM
  then for all the gears in decreasing order
  -if Wheel RPM * gear> Min Engine RPM
  -Decrement Gears

end ShiftGears

function Resetcar()
Appropriate translate and rotation transforms are used to get the new location of the car

end Resetcar

Wheel Alignment Script

function Update ()
- Define a hit point for the raycast collision
- Get the wheel collider's center point
- Cast a ray out from the wheel collider's center the distance of the suspension
- if raycast hits something
  then Get the position where the wheel hit
  else
  let the wheel be fully extended along the suspension
- Set the wheel rotation to the rotation of the collider combined with rotation around the axle, and the rotation from steering input.
- Increase the rotation value by the rotation speed (in degrees per second).
- Define a wheelhit object, to store all of the data from the wheel collider and will allow to determine the slip of the tire.
- if slip of the tire > 11.5
  then if Slip Prefab exists
  - Create an instance of it on the ground at zero rotation

end Update.

AI Script

//This script is similar to playerCar script with the only difference that AI is used instead of user input. For AI to work , a waypoint is used. The waypoint container is used to search for all the waypoints in the scene, and the current waypoint is used to determine which waypoint in the array the car is aiming for. Input steer and input torque are the values substituted out for the player input.

function GetWaypoints ()
- Get all the waypoints in the container
- for all the transforms in the waypoints
  - if waypoint not in the container
    then add it to the array of waypoints.

end GetWaypoints

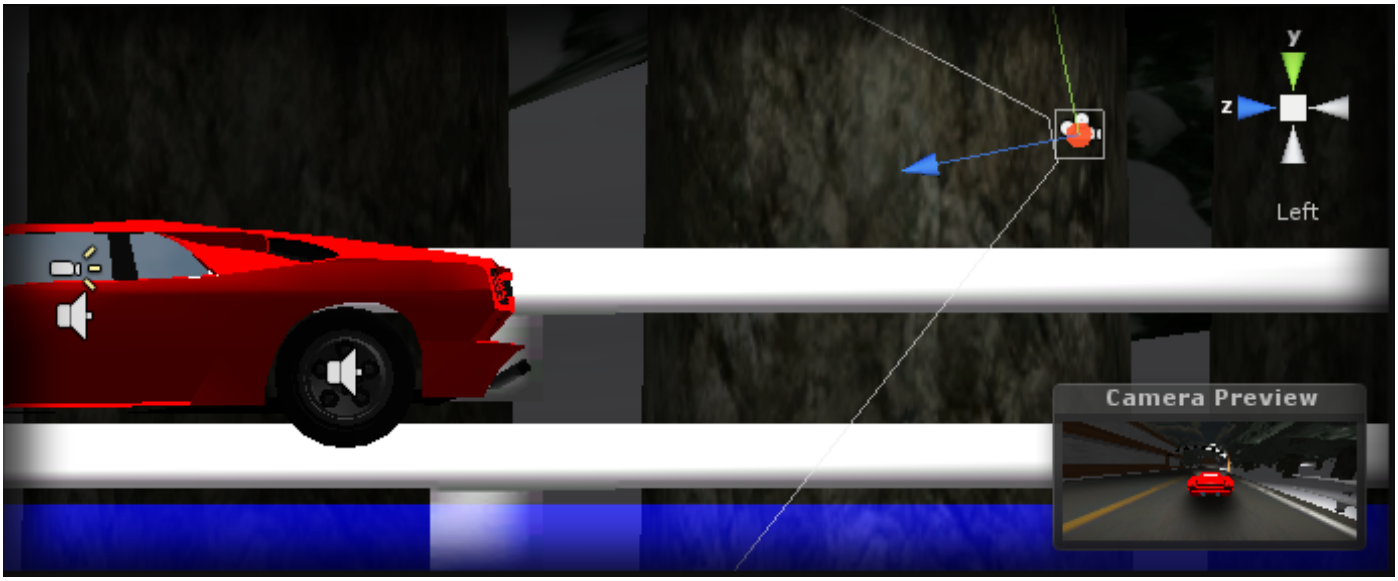
function NavigateTowardsWaypoint ()
- Get the relative position of the waypoint from the car transform to determine how far to the left and right the waypoint is
- Get a decimal percentage of the turn angle used to drive the wheels (ie) inputSteer by dividing horizontal position by the magnitude
- if (not going around a sharp turn)
  then InputTorque = horizontal position / magnitude - inputSteer
  else
  inputTorque = 0.0;
- if car position is near enough to a waypoint
  then change the target as the next waypoint
- if last waypoint is reached
  then next waypoint = first waypoint

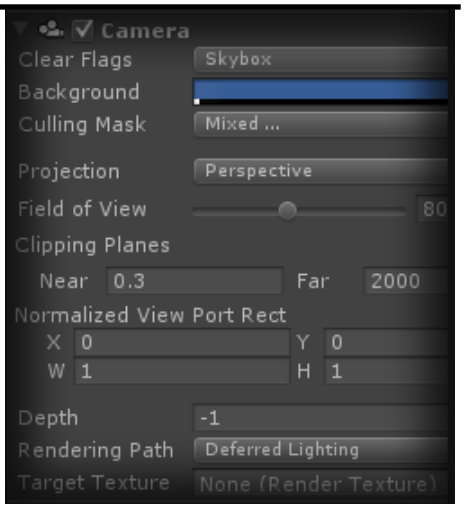
end NavigateTowardsWaypoint

4.3.4 Adding Cameras

Cameras are the devices that capture and display the world to the player. By customizing and manipulating cameras, you can make the presentation of your game truly unique. You can have an unlimited number of cameras in a scene. They can be set to render in any order, at any place on the screen, or only certain parts of the screen.

Clear Flags	Determines which parts of the screen will be cleared. This is handy when using multiple Cameras to draw different game elements.
Background	Color applied to the remaining screen after all elements in view have been drawn and there is no skybox.
Culling Mask	Include or omit layers of objects to be rendered by the Camera. Assign layers to your objects in the Inspector.
Projection	Toggles the camera's capability to simulate perspective.
Perspective	Camera will render objects with perspective intact.
Orthographic	Camera will render objects uniformly, with no sense of perspective.
Size (when Orthographic is selected)	The viewport size of the Camera when set to Orthographic.
Field of view (when Perspective is selected)	Width of the Camera's view angle, measured in degrees along the local Y axis.
Clipping Planes	Distances from the camera to start and stop rendering.
Near	The closest point relative to the camera that drawing will occur.
Far	The furthest point relative to the camera that drawing will occur.
Normalized View Port Rect	Four values that indicate where on the screen this camera view will be drawn, in Screen Coordinates (values 0-1).
X	The beginning horizontal position that the camera view will be drawn.
Y	The beginning vertical position that the camera view will be drawn.
W (Width)	Width of the camera output on the screen.
H (Height)	Height of the camera output on the screen.
Depth	The camera's position in the draw order. Cameras with a larger value will be drawn on top of cameras with a smaller value.
Rendering Path	Options for defining what rendering methods will be used by the camera.
Use Player Settings	This camera will use whichever Rendering Path is set in the Player Settings.
Target Texture (Unity Pro only)	Reference to a Render Texture that will contain the output of the Camera view. Making this reference will disable this Camera's capability to render to the screen.

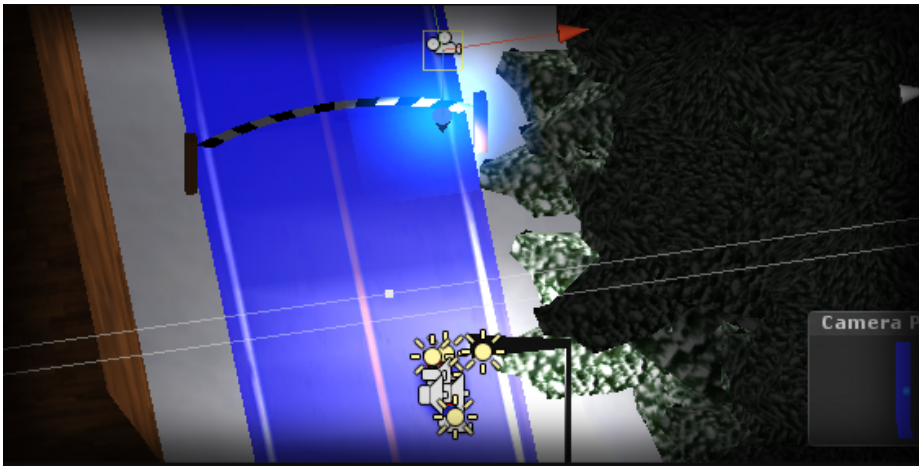


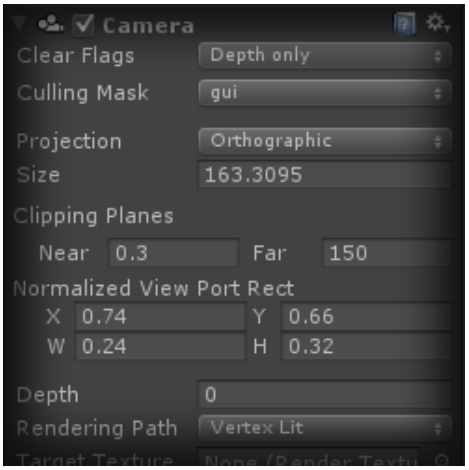


The follow camera is customized as shown. The camera preveiw shows the scene which the camera captures. The camera can be adjusted to show the field of view it can see. An important feature of the camera is the culling mask. The culling mask allows the camera to show or hide certain objects in the game.

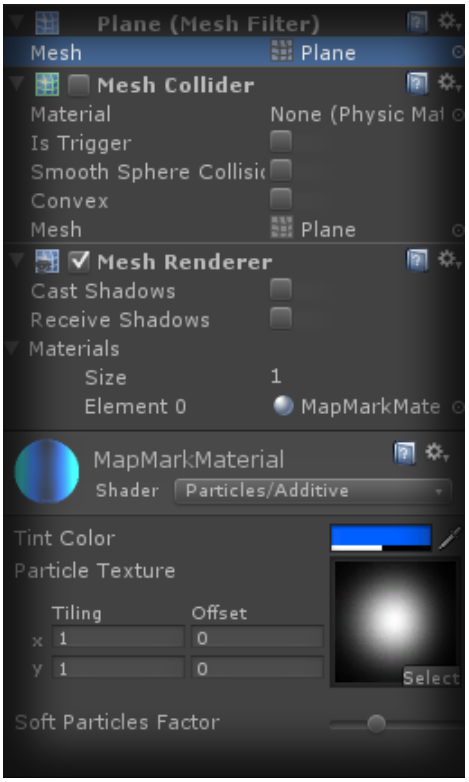
The follow camera placed behind the car shows all the objects except the racetrack which is blue in color. This blue colored racetrack is used to display the map and the position of the player car on the minimap of the game.

The main follow camera makes use of certain scripts viz. Motion Blur, Bloom And Lens Flares, Depth Of Field 34, Antialiasing As Post Effect, Edge Detect Effect, Contrast Enhance etc. These scripts change the dislay of the camera by providing certain peculiar visual effects. Either of the scripts or a combination of certain scripts can be put to use in order to attain the desired visual effect.

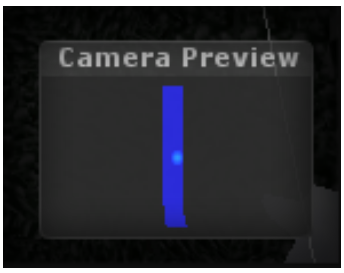




Here the camera setting (culling mask) is altered so that it displays only the blue racetrack and the map mark. The map mark is mesh plane to which a mesh collider and mesh renderer is applied. The Mesh Renderer takes the geometry from the Mesh Filter and renders it at the position defined by the object's Transform component.



The Mesh Renderer is applied the map mark material. The material has shader of the type particles/additive. The camera is placed above the map mark plane. The camera, map mark plane are placed in the body of the DiabloFX(car), due to which when the car displaces from one position to other the camera and the map mark plane also moves along with it. This is displayed as the map.



4.4 Setting up the Race Track

4.4.1 Terrain

A terrain can be created by Terrain->Create Terrain. The terrain engine gives various terrain making options: raising/lowering the height of the terrain, applying textures to the terrain surface, placing trees, stones and other objects on the terrain etc. the terrain tool provides various brushes to create or paint the terrain surface. The terrain made in the game is shown below. The terrain collider is also added.

Terrains work a bit differently than other GameObjects. You can use Brushes to paint and manipulate your Terrain. If you want to reposition a Terrain, you can modify its Transform Position values in the Inspector. This allows you to move your Terrain around, but you cannot rotate or scale it.

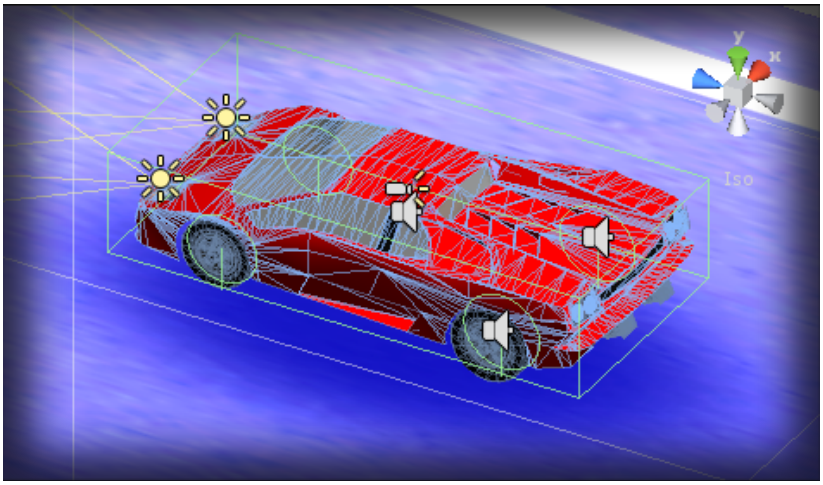
Each rectangular button is a different Terrain tool. There are tools to change the height, paint splat maps, or attach details like trees or rocks. To use a specific tool, click on it. You will then see a short description of the tool appear in text below the tool buttons.

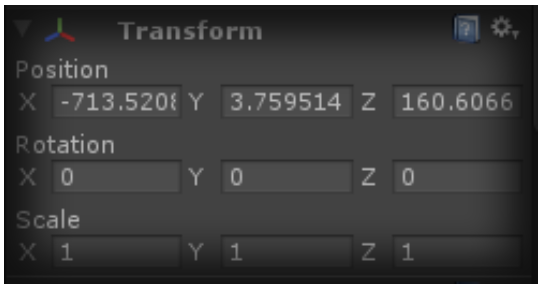
Most of the tools make use of a brush. Many different brushes are displayed for any tool that uses a brush. To select a brush, just click on it. The currently selected brush will display a preview when you hover the mouse over the terrain, at the size you have specified.

You will use all of these brushes in the Scene View to paint directly onto your Terrain. Simply choose the tool and brush you want, then click & drag on the Terrain to alter it in real-time. To paint height, textures, or decorations, you must have the Terrain selected in the Hierarchy View.

4.4.2 Placing Game Objects

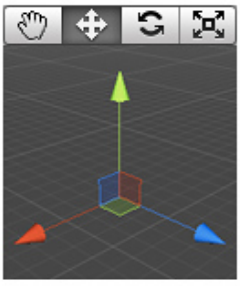
It is impossible to create a GameObject in Unity without a Transform Component. The Transform Component is one of the most important Components, since all of the GameObject's Transform properties are enabled by its use of this Component. It defines the GameObject's position, rotation, and scale in the game world/Scene View. If a GameObject did not have a Transform Component, it would be nothing more than some information in the computer's memory. It effectively would not exist in the world.



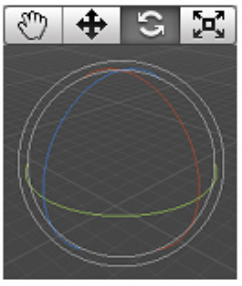


The car model displayed above. The transform component of the model shows the car position with the X,Y,Z coordinate values. These values can be changed to position the car in the scene accordingly. The rotation and the scaling of the car model can be achieved by setting the corresponding values. Transforms are always manipulated in 3D space in the X, Y, and Z axes. In Unity, these axes are represented by the colors red, green, and blue respectively. Remember: XYZ = RGB.

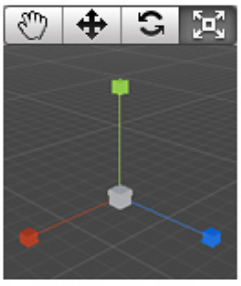
Transforms can be directly manipulated in the Scene View or by editing properties in the Inspector. In the scene, you can modify Transforms using the Move, Rotate and Scale tools. These tools are located in the upper left-hand corner of the Unity Editor.



Translate (W)



Rotate (E)

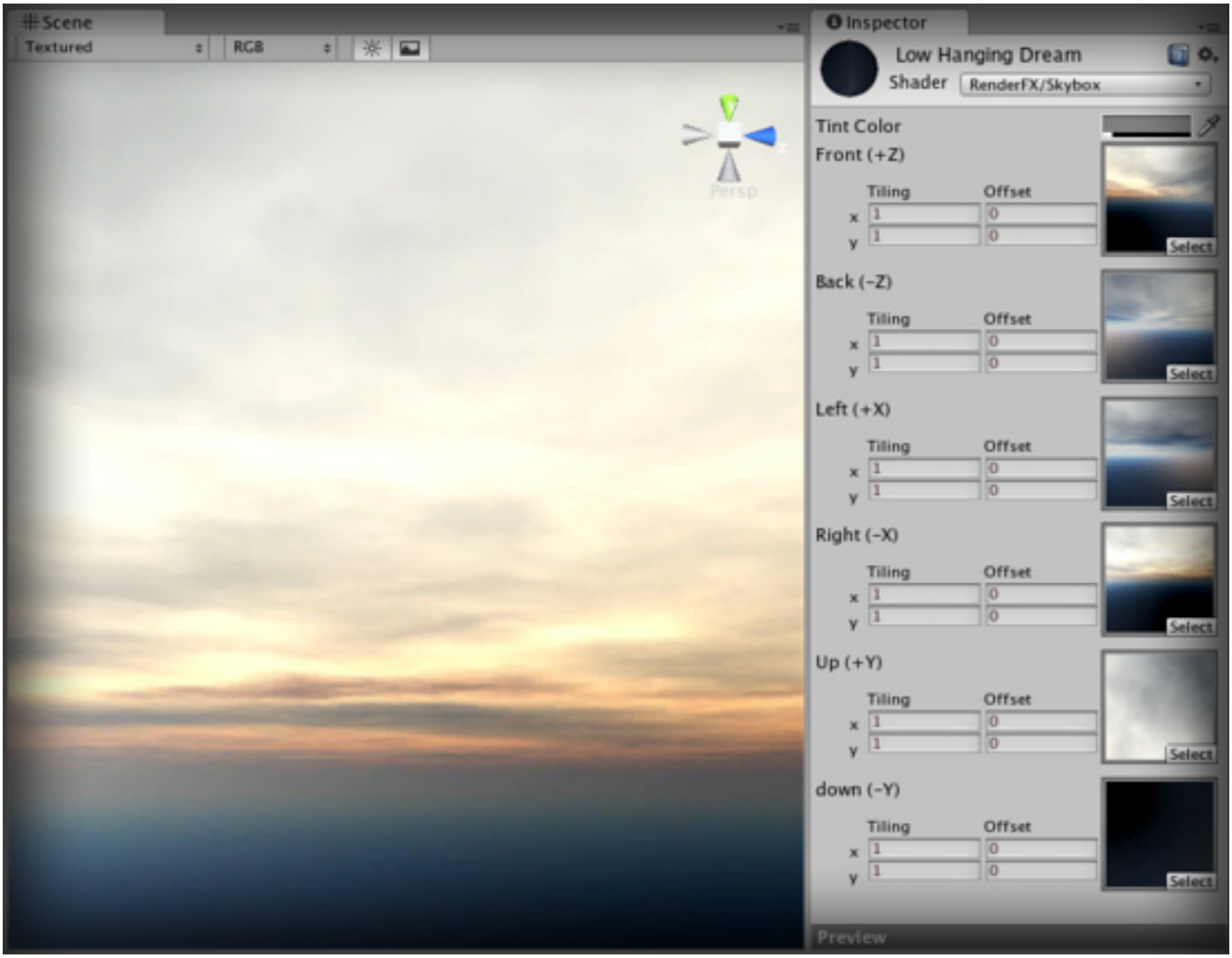


Scale (R)

4.4.3 Skybox

Skyboxes are a wrapper around your entire scene that display the vast beyond of your world.

The Material used to render the Skybox, which contains 6 Textures. This Material should use the Skybox Shader, and each of the textures should be assigned to the proper global direction.



Skyboxes are rendered before anything else in the scene in order to give the impression of complex scenery at the horizon. They are a box of 6 textures, one for each primary direction (+/-X, +/-Y, +/-Z). You have two options for implementing Skyboxes. You can add them to an individual Camera (usually the main Camera) or you can set up a default Skybox in Render Settings's Skybox Material property. The Render Settings is most useful if you want all Cameras in your scene to share the same Skybox.
Adding the Skybox Component to a Camera is useful if you want to override the default Skybox set up in the Render Settings.

4.5 Prefabs

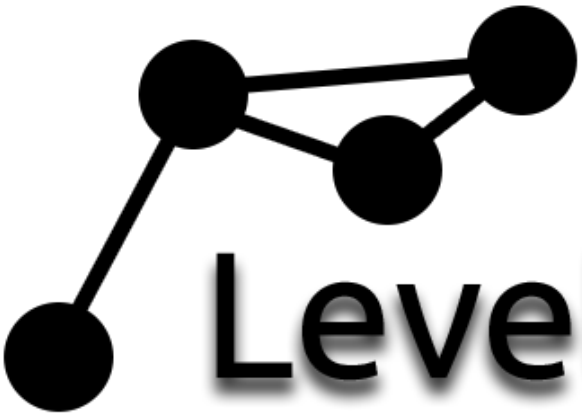
A Prefab is a type of asset -- a reusable GameObject stored in Project View. Prefabs can be inserted into any number of scenes, multiple times per scene. When you add a Prefab to a scene, you create an instance of it. All Prefab instances are linked to the original Prefab and are essentially clones of it. No matter how many instances exist in your project, when you make any changes to the Prefab you will see the change applied to all instances.

Creating Prefabs
In order to create a Prefab, simply drag a GameObject that you've created in the scene into the Project View. The GameObject's name will turn blue to show that it is a Prefab. You can rename your new Prefab.

After you have performed these steps, the GameObject and all its children have been copied into the Prefab data. The Prefab can now be re-used in multiple instances. The original GameObject in the Hierarchy has now become an instance of the Prefab.

Prefab Instances
To create a Prefab instance in the current scene, drag the Prefab from the Project View into the Scene or Hierarchy View. This instance is linked to the Prefab, as displayed by the blue text used for their name in the Hierarchy View.

Inheritance
Inheritance means that whenever the source Prefab changes, those changes are applied to all linked GameObjects. For example, if you add a new script to a Prefab, all of the linked GameObjects will instantly contain the script as well. However, it is possible to change the properties of a single instance while keeping the link intact. Simply change any property of a prefab instance, and watch as the variable name becomes bold. The variable is now overridden. All overridden properties will not be affected by changes in the source Prefab.



Chapter 5

Level Design

5.1 PlayerPrefs

PlayerPrefs stores and accesses player preferences between game sessions. It stores all the information about the player which can then be used to trigger other events.
Editor/Standalone

On Mac OS X PlayerPrefs are stored in ~/Library/Preferences folder, in a file named unity.[company name].[product name].plist, where company and product names are the names set up in Project Settings. The same .plist file is used for both Projects run in the Editor and standalone players.

On Windows, PlayerPrefs are stored in the registry under HKCU\Software\[company name]\[product name] key, where company and product names are the names set up in Project Settings.

PlayerPrefs.SetInt

static function SetInt (key : String, value : int) : void

Description
Sets the value of the preference identified by key.

```
PlayerPrefs.SetInt("Player Score", 10);
```

PlayerPrefs.GetInt

static function GetInt (key : String, defaultValue : int = 0) : int

Description
Returns the value corresponding to key in the preference file if it exists.
If it doesn't exist, it will return defaultValue.

```
print(PlayerPrefs.GetInt("Player Score"));
```

PlayerPrefs.SetFloat

static function SetFloat (key : String, value : float) : void
Description
Sets the value of the preference identified by key.

```
PlayerPrefs.SetFloat("Player Score", 10.0);
```

PlayerPrefs.GetFloat

static function GetFloat (key : String, defaultValue : float = 0.0F) : float
Description
Returns the value corresponding to key in the preference file if it exists.
If it doesn't exist, it will return defaultValue.

```
print (PlayerPrefs.GetFloat("Player Score"));
```

PlayerPrefs.SetString

static function SetString (key : String, value : String) : void

Description
Sets the value of the preference identified by key.

```
PlayerPrefs.SetString("Player Name", "Foobar");
```

PlayerPrefs.GetString

static function GetString (key : String, defaultValue : String = "") : String

Description
Returns the value corresponding to key in the preference file if it exists.
If it doesn't exist, it will return defaultValue.

```
print (PlayerPrefs.GetString("Player Name"));
```

PlayerPrefs.HasKey

static function HasKey (key : String) : boolean

Description
Returns true if key exists in the preferences.

PlayerPrefs.DeleteKey

static function DeleteKey (key : String) : void

Description
Removes key and its corresponding value from the preferences.
PlayerPrefs.DeleteAll
static function DeleteAll () : void
Description
Removes all keys and values from the preferences. Use with caution.

PlayerPrefs.Save

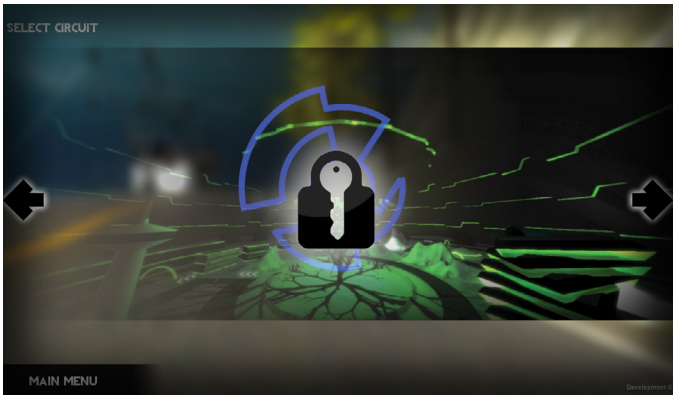
static function Save () : void

Description
Writes all modified preferences to disk.

By default Unity writes preferences to disk on Application Quit. In case when the game crashes or otherwise prematuraly exits, you might want to write the PlayerPrefs at sensible 'checkpoints' in your game. This function will write to disk potentially causing a small hiccup, therefore it is not recommended to call during actual gameplay.

5.2 Unlocking System

Our project consists of an unlocking system. In which all the cars and levels except the first ones are locked. So that as a player proceeds in the game and achieves certain goals , only then the levels and the cars can be unlocked.

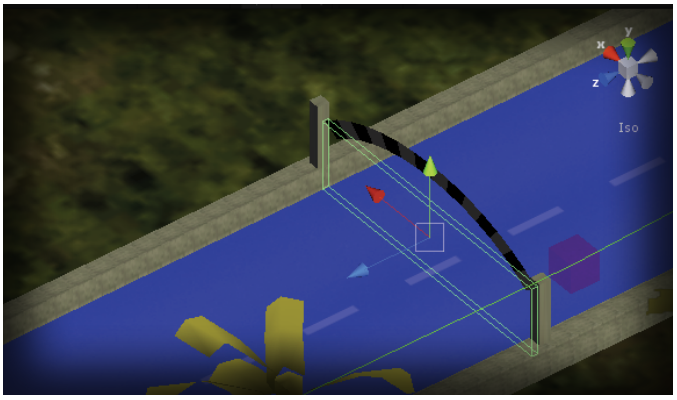


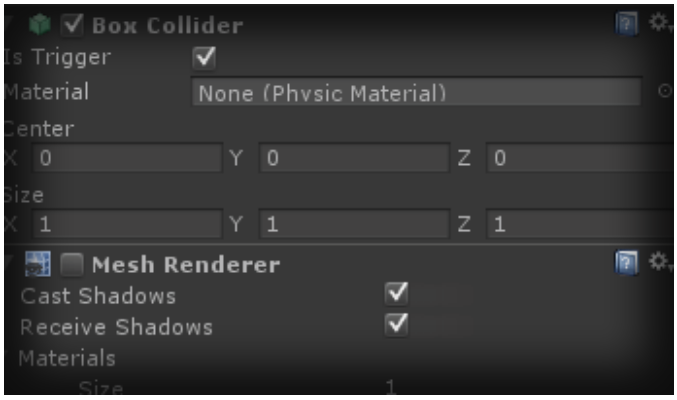
There are two types of unlocking systems in our project :

1) Level Unlocking System
2) Car Unlocking System

In Unlocking System , a box collider is placed on the track with its isTrigger condition on. It means that when an object collides with it, an event is triggered. The target object is assigned to the car body so that when a car collides with the box collider ,it triggers an event which helps in unlocking the next level or car with the use of PlayerPrefs.

If the player is in the quick race mode , then the next **Track** is unlocked
if the player is in the time trial mode , then the next **Car** is unlocked.



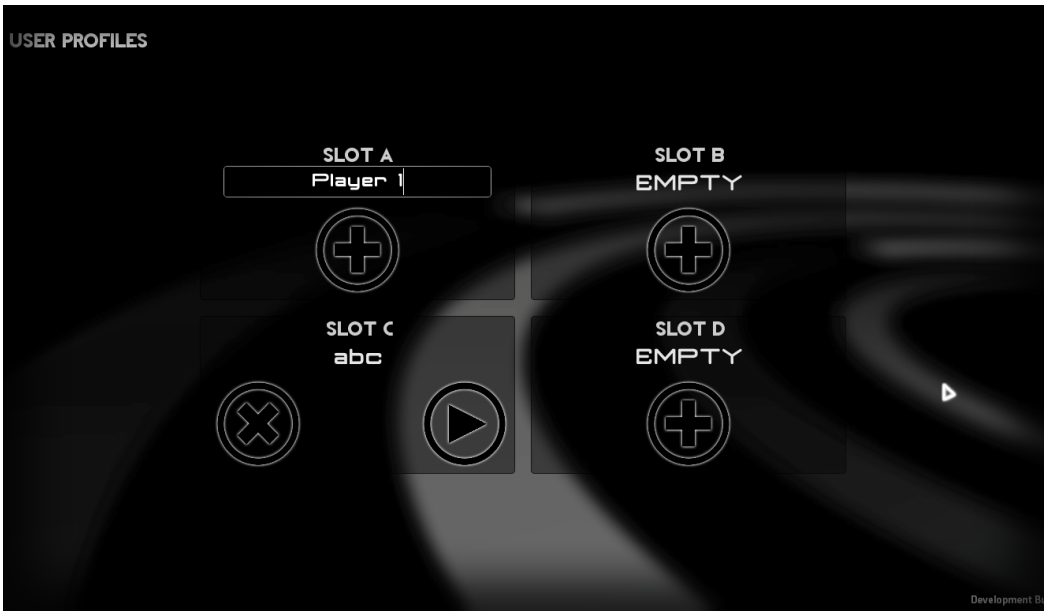


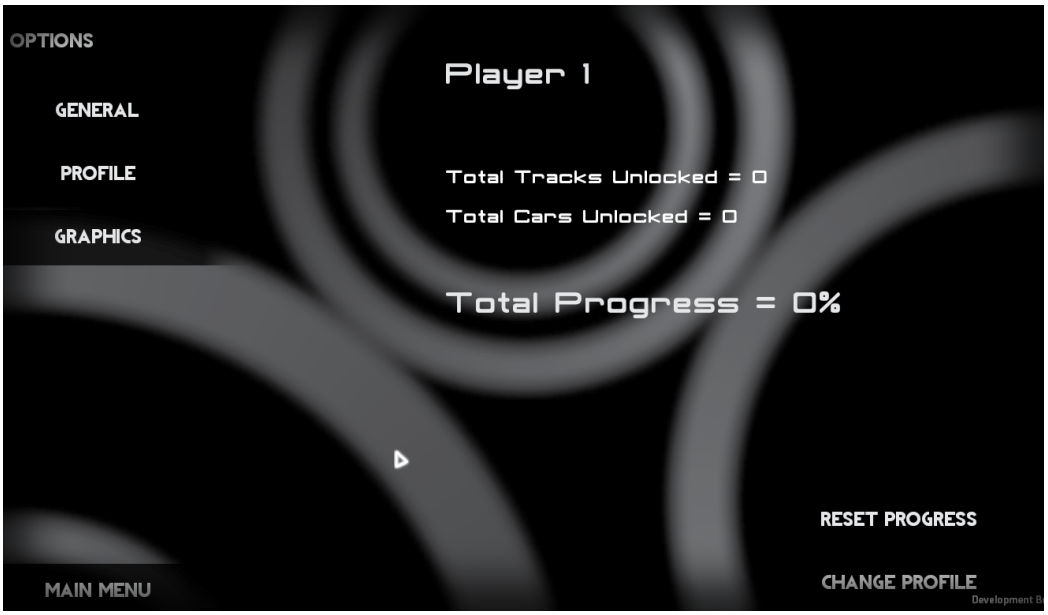
5.3 User Profiles

Our project supports upto four different profiles. A new profile can be created in a new slot. Initially all slots are empty. While creating the profile the player can give a name to the profile. Once a player creates a new profile, the progress is reset to the default value(0) , all the information about the player like the name and progress will be saved in a set of PlayerPrefs.

As the player progresses through the levels, the profile progress gets updated, so that when a player plays later with the same profile, the previous data about the player will be restored and the progress can be resumed. Progress corresponds to the number of levels unlocked and number of cars unlocked. An active profile can be deleted. On deletion all progress will be lost permanently.

Profile progress details can be viewed in the options menu. Players can even switch profiles from the options menu.





5.2 Game Modes

5.2.1 Single Player

Quick Race

In Quick Race mode, a player has three AI opponents to compete with. The player is allowed to choose the number of laps. The mechanism of laps is as follows:

There are three planes kept across the track, Checkpoint 1 (CPT1) , Checkpoint 2 (CPT2), and Start-finish (SF). When a player car goes from SF to CPT1 a variable "cpt" is incremented from 0 to 1. From CPT1 to CPT2 "cpt" is incremented from 1 to 2 and From CPT2 to SF "cpt" is changed from 2 to 0 and only then the lap counter is incremented and not otherwise. This ensures that the player has to go around the entire track if his lap counter needs to be incremented. Also when the required number of laps get over , the race finishes and the position of the player with respect to the opponents is saved. The time taken by the player is also saved.

The functionality to keep the track of the position of the player is as follows: There are different set of planes kept on the track in groups of three ie. p0,p1,p2. When the player goes from p0 to p1 , p1 to p2, or p2 to p0 then the position counter is incremented and when the car goes from p2 to p1 , p1 to p0 or p0 to p2 (ie) the car starts going in the opposite direction, the position counter is decremented and it is indicated to the player that he is going in the wrong direction by a signal.

The position counters of the three opponents and the player are compared by the controller script to keep the track of their position in the race with respect to others.

A timer is kept to record the current lap time of a player in the race and the timer goes on incrementing registering a lap time till every lap is completed and the race finishes . if a player finishes the race in position 1, then the player wins. The next level is unlocked only the player wins in the current level.

Time Trial Mode

In Time Trial mode, there is a different predefined timer which tells in how much time 1 lap of the track has to be completed in. The timer is is decremented every second. The mechanism of timer is as follows:There is a StartTime which has the value of the time at the start of the race , then the current time in the game is calculated and is kept on subtracting from the StartTime. The time remaining is displayed on the screen. Once the time remaining becomes 0, and if the player has not reached the finish line then a message is displayed saying that you have not completed the lap in time and the race is ended.In time trial, if the player finishes a race before the timer ends, then a new car maybe unlocked.

5.2.2 Multi Player

A multiplayer video game is a video game in which more than one person can play in the same game environment at the same time. Video games are often single-player activities that put the player against pre-programmed challenges and/or AI-controlled opponents, which often lack the flexibility and ingenuity of regular human thinking.

Multiplayers components allow players to enjoy interaction with other individuals, be it in the form of partnership, competition or rivalry. Furthermore, it provides them with a form of social communication that is almost always missing in single-player oriented games. In a variety of different multiplayer game types, players may individually compete against two or more human contestants, work cooperatively with a human partner(s) in order to achieve a common goal, supervise activities of other players, or engage in a game type that incorporates any possible combination of the above.

Split - Screen

Sometimes, you want to want to play games with a real person who's sitting right next to you. Thats why we have included Split-Screen Multiplayer in our game. This allows two players to simultaneously compete against each other on the same PC.

This is Accomplished by accomodating two cameras in the viewport so that 1 camera is dedicated for each player. Each camera has its own Heads Up display corresponding to that player. Another Major Difference is the input method for the two players. Player 1 controls are assigned as the Arrow Keys while Player 2 is Assigned the WASD key controls. We can define seperate horizontal and vertical axes for each players for each player.

The Gameplay is similar to the quick race mode in which each player has to complete a predefined number of laps, and the winner is decided based on who crosses the finish line first.

Lan Play

One of the most exciting features in any video game is playing against your own friends. Unity provides a basic networking framework that can be used to connect various instances of the game running on different machines.

However a more advanced knowledge of networking is essential in making sure any video game runs bug free. There are various hurdles that we might face while we implement the multiplayer via a Local Area Network. The most basic issue is connectivity. We have to make sure we get rid of all the hurdles that might prevent a basic connection. Firewalls in particular are to be disabled. Also we need to make sure file sharing is enabled in the control panel and all participating machines belong to the same workgroup. Once the connection is established, our next issue is latency. In simple words, higher latency results in a more severe lag. This could be either due to noisy hardware connections or inefficient state synchronization.

Networking is communication between two or more computers. A fundamental idea is that of the relationship between the client (the computer that is requesting information) and the server (the computer responding to the information request). The server can either be a dedicated host machine used by all clients, or simply a player machine running the game (client) but also acting as the server for other players. Once a server has been established and a client has connected to it, the two computers can exchange data as demanded by gameplay.

There are two common and proven approaches to structuring a network game which are known as Authoritative Server and Non-Authoritative Server. The authoritative server approach requires the server to perform all world simulation, application of game rules and processing of input from the player clients. Each client sends their input (in the form of keystrokes or requested actions) to the server and continuously receives the current state of the game from the server. The client never makes any changes to the game state itself. Instead, it tells the server what it wants to do, and the server then handles the request and replies to the client to explain what happened as a result. A non-authoritative server does not control the outcome of every user input. The clients themselves process user input and game logic locally, then send the result of any determined actions to the server. The server then synchronizes all actions with the world state.

Remote Procedure Calls (RPCs) are used to invoke functions on other computers across the network, although the "network" can also mean the message channel between the client and server when they are both running on the same computer. To create the server, you simply call Network.InitializeServer() from a script. When you want to connect to an existing server as a client, you call Network.Connect() instead. The Network View is a Component that sends data across the network. Network Views make your GameObject capable of sending data using RPC calls or State Synchronization. The way you use Network Views will determine how your game's networking behaviors will work. Network Views have few options, but they are incredibly important for your networked game.

Network.Instantiate() lets you instantiate a prefab on all clients in a natural and easy way. Essentially this is an Instantiate() call, but it performs the instantiation on all clients. The Master Server helps you match games. When you start a server you connect to the master server, and it provides a list of all the active servers.

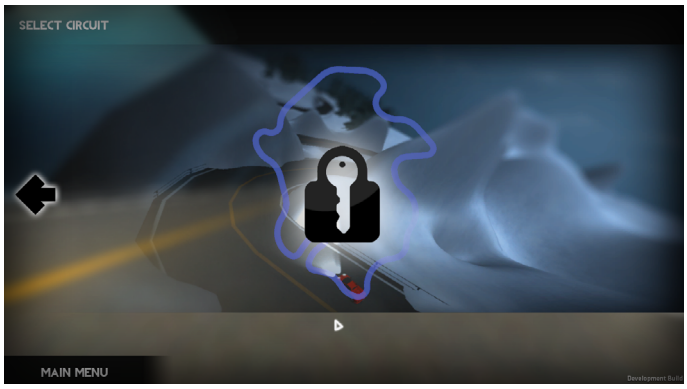
5.3 Car & Track Selection Menu

4.4.1 Track Selection Menu

On the track selection menu, the player can choose from the tracks which are unlocked using next and previous arrow buttons. These two buttons keep the track of how many times they have been pressed and in what sequence to determine which track has been chosen. On the track selection menu, there is a button “SELECT CAR” to go to the car selection menu. Once this button is clicked, the identifier of the chosen track is saved into a player pref which is then used to load the corresponding level when the race is started.

For quick race mode, track will get unlocked as the player clears each level. Also the number of laps can be chosen by the player. In time trial mode, all the tracks which are unlocked in the quick race mode will be unlocked.

For split screen mode, the tracks which have been unlocked by the player in the quick race mode will be unlocked.



4.4.2 Car Selection Menu

Car selection menus are different for single player and multiplayer.

For single player, the player can choose from the cars which are unlocked using next and previous arrow buttons. These two buttons keep the track of how many times they have been pressed and in what sequence to determine which car has been chosen. Also the colour of the car can be changed by choosing the colours from the menu. Once a car model and its colour has been chosen , the button “RACE” is clicked , at which point the identifier of the chosen car is saved into a player pref which is then used to instantiate the car model on the track at a fixed position when the race is started.

For multiplayer car selection menu , the screen is again divided into two parts. The first part contains the profile of the player which was loaded initially , the player can choose the car and the colour and get ready for the race. The second part gives the player a choice of profiles to choose from. Once a profile is chosen , then the car model and the colour can be selected. When both the players are ready for the race and press the “RACE” button, the race starts instantiating the respective selected cars of the two players.

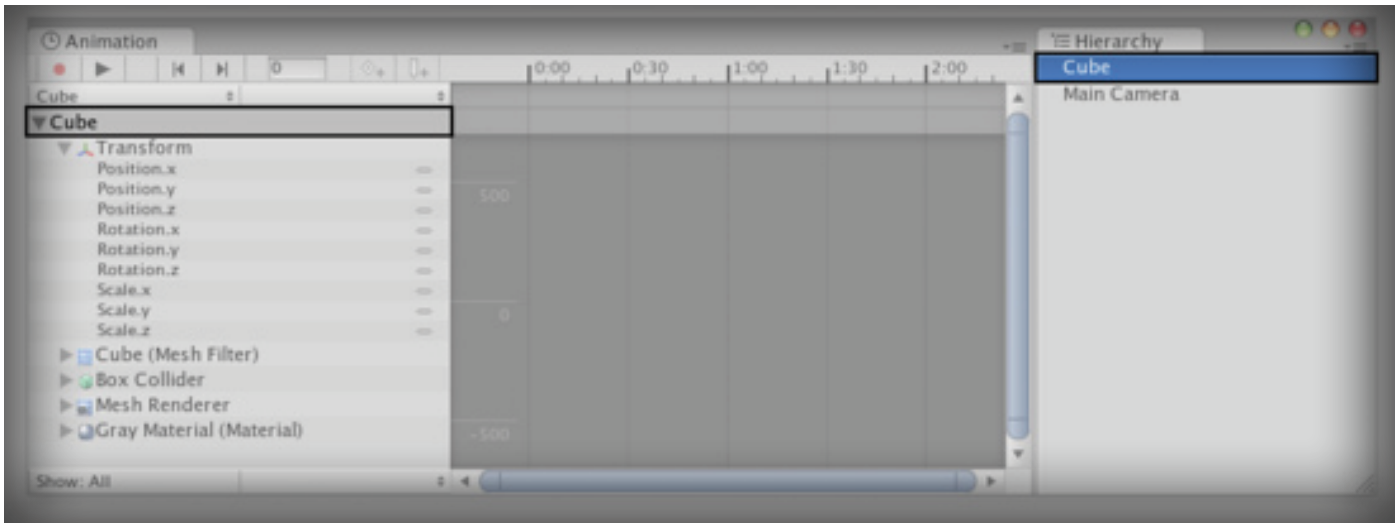
For split screen mode, the cars which have been unlocked by the player in the time trial mode will be unlocked.

Chapter 6

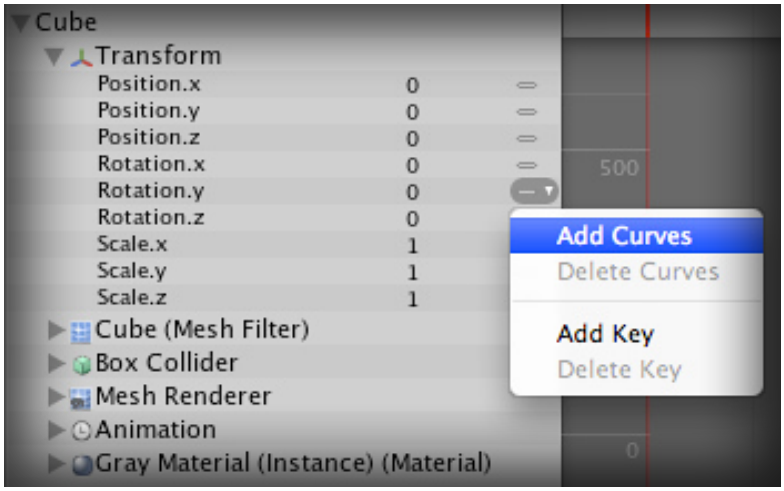
Tweaking

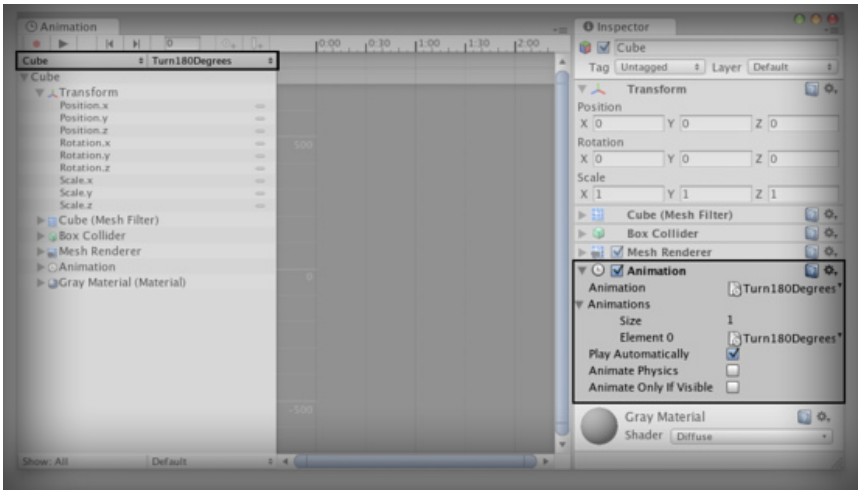
6.1 Animation

The Animation View in Unity allows you to create and modify Animation Clips directly inside Unity. It is designed to act as a powerful and straightforward alternative to external 3D animation programs. In addition to animating movement, the editor also allows you to animate variables of materials and components and augment your Animation Clips with Animation Events, functions that are called at specified points along the timeline.



The Animation View is tightly integrated with the Hierarchy View, the Scene View, and the Inspector. Like the Inspector, the Animation View will show whatever Game Object is selected. You can select a Game Object to view using the Hierarchy View or the Scene View. (If you select a Prefab in the Project View you can inspect its animation curves as well, but you have to drag the Prefab into the Scene View in order to be able to edit the curves. At the left side of the Animation View is a





hierarchical list of the animatable properties of the selected Game Object. The list is ordered by the Components and Materials attached to the Game Object, just like in the Inspector. Components and Materials can be folded and unfolded by clicking the small triangles next to them. If the selected Game Object has any child Game Objects, these will be shown after all of the Components and Materials.

Creating a New Animation Clip

Animated Game Objects in Unity need an Animation Component that controls the animations. If a Game Object doesn't already have an Animation Component, the Animation View can add one for you automatically when creating a new Animation Clip or when entering Animation Mode.

To create a new Animation Clip for the selected Game Object, click the right of the two selection boxes at the upper right of the Animation View and select [Create New Clip]. You will then be prompted to save an Animation Clip somewhere in your Assets folder. If the Game Object doesn't have an Animation Component already, it will be automatically added at this point. The new Animation Clip will automatically be added to the list of Animations in the Animation Component.

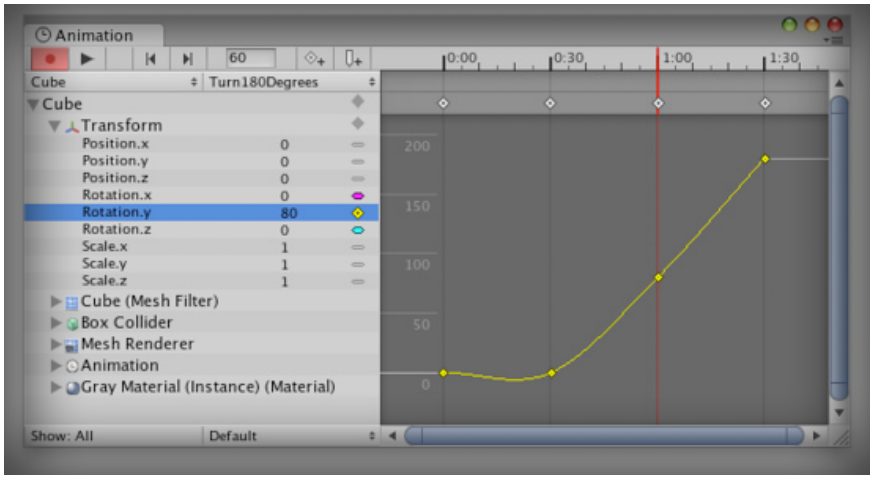
Animating a Game Object

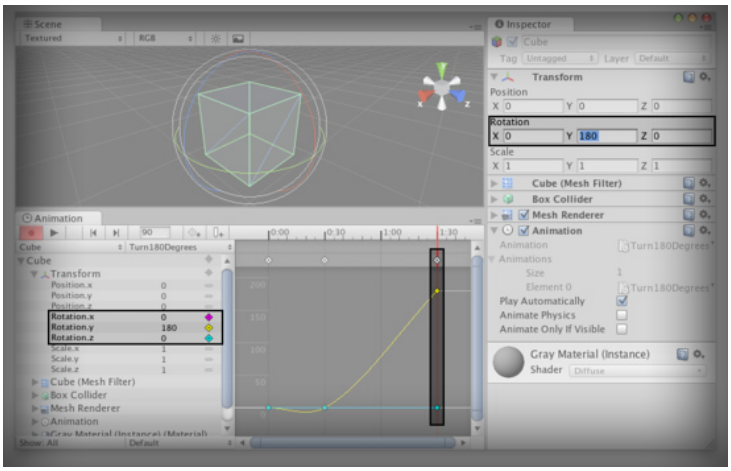
To begin editing an Animation Clip for the selected Game Object, click on the Animation Mode button.

This will enter Animation Mode, where changes to the Game Object are stored into the Animation Clip. (If the Game Object doesn't have an Animation Component already, it will be automatically added at this point. If there is not an existing Animation Clip, you will be prompted to save one somewhere in your Assets folder.)

You can stop the Animation Mode at any time by clicking the Animation Mode button again. This will revert the Game Object to the state it was in prior to entering the Animation Mode.

You can animate any of the properties shown in the property list of the Animation View. To animate a property, click on the Key Indicator for that property and choose Add Curve from the menu. You can also select multiple properties and right click on the selection to add curves for all the selected properties at once.



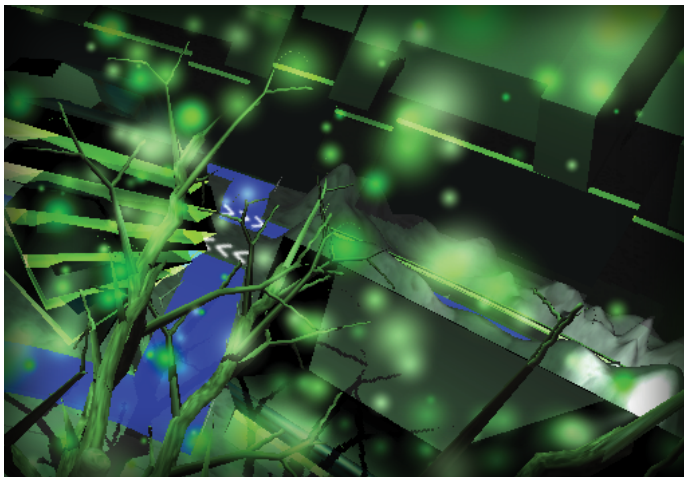


You can also manually create a keyframe using the Keyframe button. This will create a key for all the curves that are currently shown in the Animation View. If you want to only show curves for a subset of the properties in the property list, you can select those properties. This is useful for selectively adding keys to specific properties only

6.2 Particle Systems

Particles are essentially 2D images rendered in 3D space. They are primarily used for effects such as smoke, fire, water droplets, or leaves. A Particle System is made up of three separate Components: Particle Emitter, Particle Animator, and a Particle Renderer. You can use a Particle Emitter and Renderer together if you want static particles. The Particle Animator will move particles in different directions and change colors. You also have access to each individual particle in a particle system via scripting, so you can create your own unique behaviors that way if you choose.

Ellipsoid Particle Emitters (EPEs) are the basic emitter, and are included when you choose to add a Particle System to your scene from Components->Particles->Particle System. You can define the boundaries for the particles to be spawned, and give the particles an initial velocity. From here, use the Particle Animator to manipulate how your particles will change over time to achieve interesting effects. Particle Emitters work in conjunction with Particle Animators and Particle Renderers to create, manipulate, and display Particle Systems. All three Components must be present on an object before the particles will behave correctly. When particles are being emitted, all different velocities are added together to create the final velocity. Spawning properties like Size, Energy, Emission, and Velocity will give your particle system distinct personality when trying to achieve different effects.Energy and Emission will control how long your particles remain onscreen and how many particles can appear at any one time. For example, a rocket might have high Emission to simulate density of smoke, and high Energy to simulate the slow dispersion of smoke into the air.



6.3 Heads up Display

A head-up display or heads-up display—also known as a HUD—is any transparent display that presents data without requiring users to look away from their usual viewpoints. The origin of the name stems from a pilot being able to view information with the head positioned "up" and looking forward, instead of angled down looking at lower instruments. In video gaming, the HUD (heads-up display) is the method by which information is visually relayed to the player as part of a game's user interface. It takes its name from the head-up displays used in modern aircraft.The HUD is frequently used to simultaneously display several pieces of information including the main character's health, items, and an indication of game progression (such as score or level).

UnityGUI allows you to create a wide variety of highly functional GUIs very quickly and easily. Rather than creating a GUI object, manually positioning it, and then writing a script that handles its functionality, you can do everything at once with just a few lines of code. The code produces GUI controls that are instantiated, positioned and handled with a single function call.

Heads- Up Display Elements can be esaily integrated using UnityGUI. The essential part here is that it is linked directly to the game statistics. Hence it does not remain static and constantly changes as the game progresses. We have designed different HUD's for each Game Mode.

6.4 GUI Skins / Components

GUISkins are a collection of GUIStyles that can be applied to your GUI. Each Control type has its own Style definition. Skins are intended to allow you to apply style to an entire UI, instead of a single Control by itself.

In some cases you want to have two of the same Control with different Styles. For this, it does not make sense to create a new Skin and re-assign it. Instead, you use one of the Custom Styles in the skin. Provide a Name for the custom Style, and you can use that name as the last argument of the individual Control.

When you are creating an entire GUI for your game, you will likely need to do a lot of customization for every different Control type. In many different game genres, like real-time strategy or role-playing, there is a need for practically every single Control type.

Because each individual Control uses a particular Style, it does not make sense to create a dozen-plus individual Styles and assign them all manually. GUI Skins take care of this problem for you. By creating a GUI Skin, you have a pre-defined collection of Styles for every individual Control. You then apply the Skin with a single line of code, which eliminates the need to manually specify the Style of each individual Control.

Button

```
if (GUI.Button (Rect (10,10,150,100), "I am a button")) {
        print ("You clicked the button!");
}
```

Label

```
GUI.Label (Rect (0,0,100,50), "This is the text string for a Label Control");
```

Box

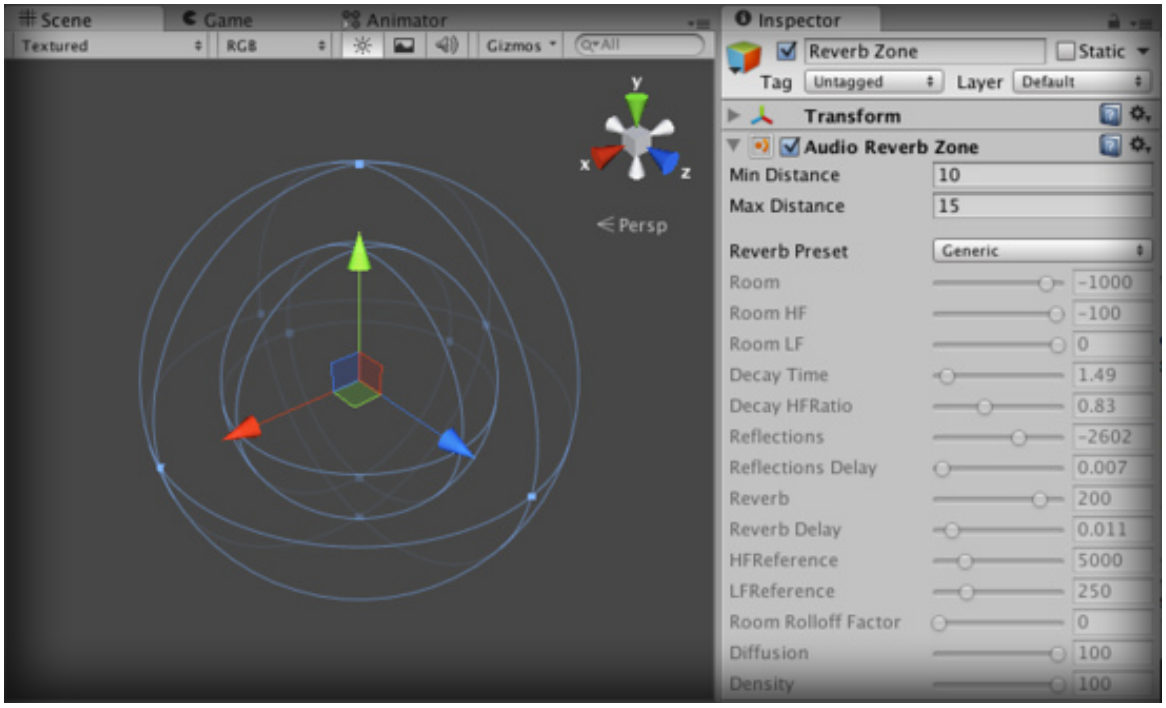
```
GUI.Box(new Rect(10,10,100,90), "This is the text string for a Box");
```

TextField

```
textFieldString = GUI.TextField (Rect (25, 25, 100, 30), textFieldString);
```

6.5 Audio Reverb Zones

Reverb Zones take an Audio Clip and distort it depending where the audio listener is located inside the reverb zone. They are used when you want to gradually change from a point where there is no ambient effect to a place where there is one, for example when you are entering a cavern.

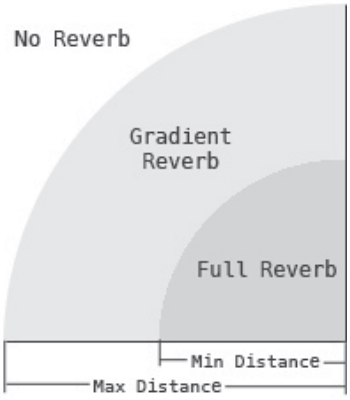


Properties

Min Distance	Represents the radius of the inner circle in the gizmo, this determines the zone where there is a gradually reverb effect and a full reverb zone.
Max Distance	Represents the radius of the outer circle in the gizmo, this determines the zone where there is no effect and where the reverb starts to get applied gradually.
Reverb Preset	Determines the reverb effect that will be used by the reverb zone.

Adding a Reverb Zone

To add a Reverb Zone to a given audio source just select the object in the inspector and then select Component->Audio->Audio Reverb Zone.



6.6 Image Effects

This group handles all Render Texture-based fullscreen image postprocessing effects. They are only available with Unity Pro. They add a lot to the look and feel of your game without spending much time on artwork.All image effects make use of Unity's OnRenderImage function which any MonoBehavior attached to a camera can overwrite to accomplish a wide range of custom effects. Here are some examples:-

Antialiasing (PostEffect) offers a set of algorithms designed to give a smoother appearance to graphics. When two areas of different colour adjoin in an image, the shape of the pixels can form a very distinctive "staircase" along the boundary. This effect is known as aliasing and hence antialiasing refers to any measure which reduces the effect.

Bloom and Lens Flares image effect adds bloom and also automatically generates lens flares in a highly efficient way. Bloom is a very distinctive effect that can make a big difference to a scene and may suggest a magical or dreamlike environment especially when used in conjunction with HDR rendering.

Motion Blur image effect enhances fast-moving scenes by leaving "motion trails" of previously rendered frames. Like all image effects, Motion Blur is only available in Unity Pro.

Blur image effect blurs the rendered image in real-time.

Contrast Stretch dynamically adjusts the contrast of the image according to the range of brightness levels it contains. The adjustment takes place gradually over a period of time, so the player can be briefly dazzled by bright outdoor light when emerging from a dark tunnel.

Screen Space Ambient Occlusion (SSAO) image effect approximates Ambient Occlusion in realtime, as an image post-processing effect. It darkens creases, holes and surfaces that are close to each other. In real life, such areas tend to block out or occlude ambient light, hence they appear darker.

Depth of Field 3.4 is a common postprocessing effect that simulates the properties of a camera lens. a camera can only focus sharply on an object at a specific distance.

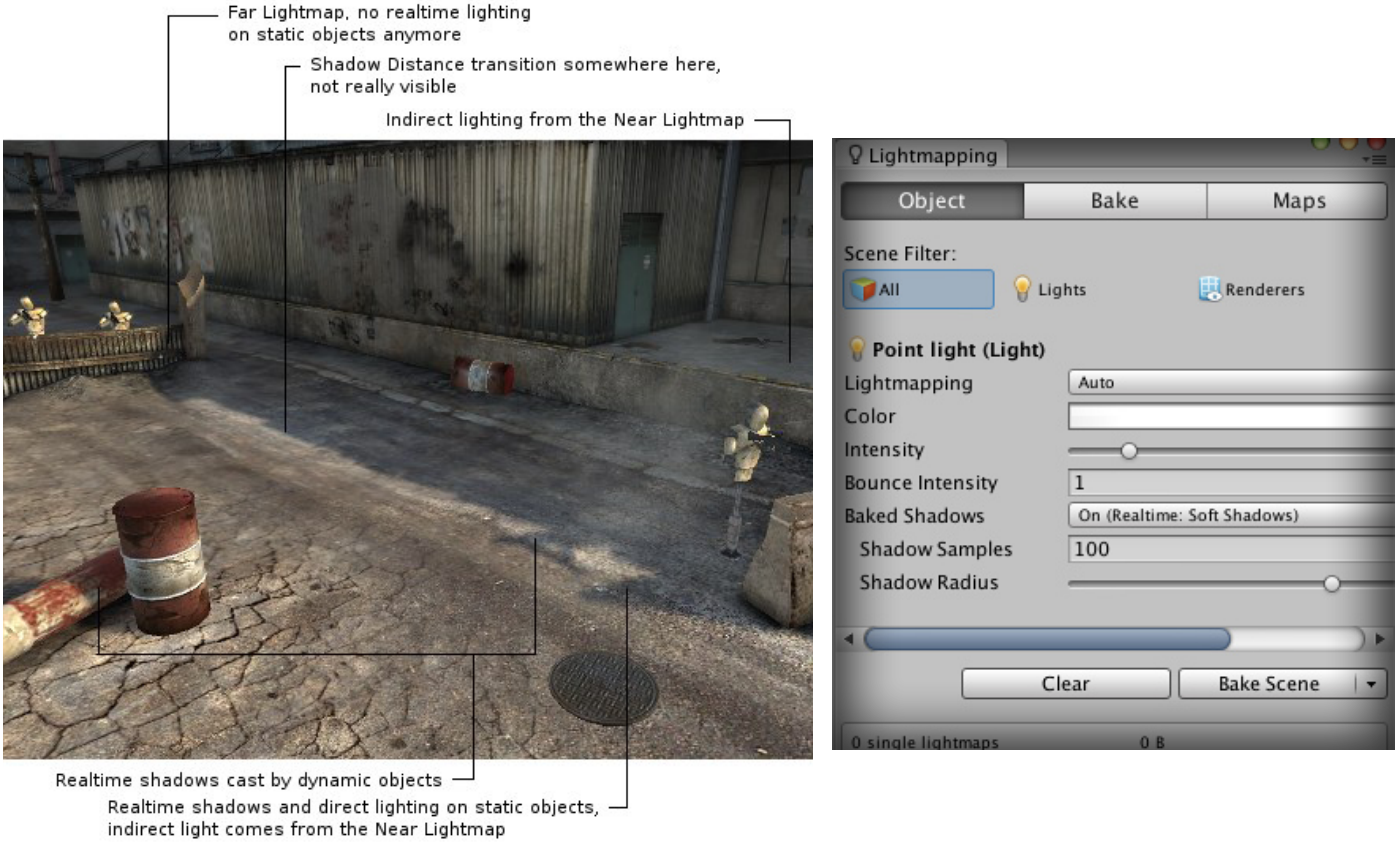
6.7 Light - Mapping

Unity has a fully integrated lightmapper – Beast by Illuminate Labs. This means that Beast will bake lightmaps for your scene based on how your scene is set up within Unity, taking into account meshes, materials, textures and lights. It also means that lightmapping is an integral part of the rendering engine - once your lightmaps are created you don't need to do anything else, they will be automatically picked up by the objects.

With Unity Pro we can take the scene one step further by enabling Global Illumination and adding a Sky Light. In the Bake pane we set the number of Bounces to 1 and the Sky Light Intensity to 0.5. The result is much softer lighting with subtle diffuse interreflection effects (color bleeding from the green and blue cubes) - much nicer and it's still only 3 cubes and a light!

Lightmapping is fully integrated in Unity, so that you can build entire levels from within the Editor, lightmap them and have your materials automatically pick up the lightmaps without you having to worry about it. Lightmapping in Unity means that all your lights' properties will be mapped directly to the Beast lightmapper and baked into textures for great performance. Unity Pro extends this functionality by Global Illumination, allowing for baking realistic and beautiful lighting, that would otherwise be impossible in realtime. Additionally Unity Pro brings you sky lights and emissive materials for even more interesting scene lighting.

Dual lightmaps is Unity's approach to make lightmapping work with specular, normal mapping and proper blending of baked and realtime shadows. It's also a way to make your lightmaps look good even if the lightmap resolution is low. Dual lightmaps by default can only be used in the Deferred Lighting rendering path. In Forward rendering path, it's possible to enable Dual Lightmaps by writing custom shaders (use dualforward surface shader directive).



Chapter 7

Deployment & Maintainance

7.1 Project Settings

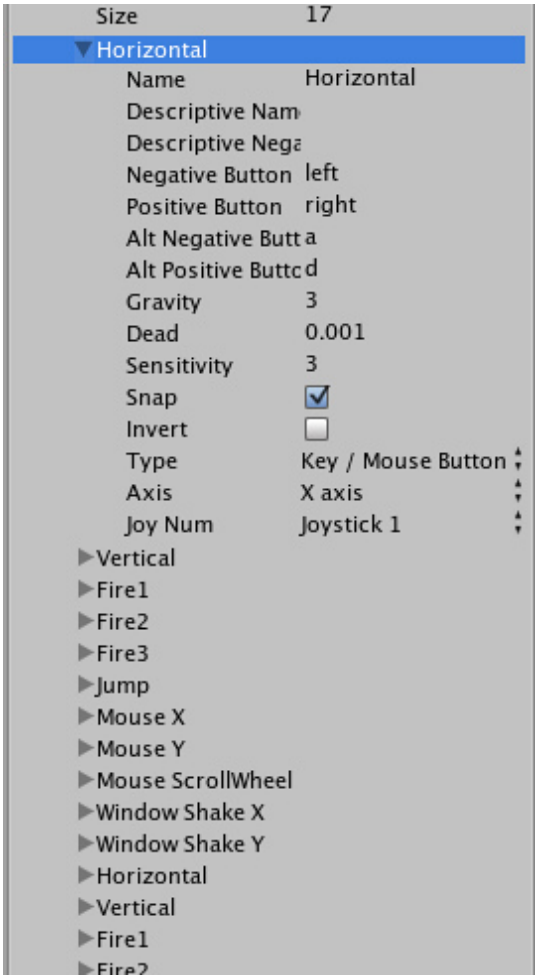
7.1.1 Input Settings

Adding new Input Axes

To add a new virtual axes go to the Edit->Project Settings->Input menu. Here the settings of each axis can be changed.

Name	The name of the string used to check this axis from a script.
Descriptive Name	Positive value name displayed in the input tab of the Configuration dialog for standalone builds.
Descriptive Negative Name	Negative value name displayed in the Input tab of the Configuration dialog for standalone builds.
Negative Button	The button used to push the axis in the negative direction.
Positive Button	The button used to push the axis in the positive direction.
Alt Negative Button	Alternative button used to push the axis in the negative direction.
Alt Positive Button	Alternative button used to push the axis in the positive direction.
Gravity	Speed in units per second that the axis falls toward neutral when no buttons are pressed.
Dead	Size of the analog dead zone. All analog device values within this range result map to neutral.
Sensitivity	Speed in units per second that the the axis will move toward the target value. This is for digital devices only.
Snap	If enabled, the axis value will reset to zero when pressing a button of the opposite direction.
Invert	If enabled, the Negative Buttons provide a positive value, and vice-versa.
Type	The type of inputs that will control this axis.
Axis	The axis of a connected device that will control this axis.
Joy Num	The connected Joystick that will control this axis.

These settings can be used to fine tune the look and feel of input. They are all documented with tooltips in the Editor as well.



Using Input Axes from Scripts
The current state from a script can be queried like this:
value = Input.GetAxis ("Horizontal");
An axis has a value between -1 and 1. The neutral position is 0. This is the case for joystick input and keyboard input.
However, Mouse Delta and Window Shake Delta are how much the mouse or window moved during the last frame. This means it can be larger than 1 or smaller than -1 when the user moves the mouse quickly.
It is possible to create multiple axes with the same name. When getting the input axis, the axis with the largest absolute value will be returned. This makes it possible to assign more than one input device to one axis name. For example, create one axis for keyboard input and one axis for joystick input with the same name. If the user is using the joystick, input will come from the joystick, otherwise input will come from the keyboard. This way you don't have to consider where the input comes from when writing scripts.

7.1.2 Quality Settings

Unity allows you to set the level of graphical quality it will attempt to render. Generally speaking, quality comes at the expense of framerate and so it may be best not to aim for the highest quality on mobile devices or older hardware since it will have a detrimental effect on gameplay. The Quality Settings inspector (menu: Edit->Project Settings->Quality) is split into two main areas.
Unity lets you assign a name to a given combination of quality options for easy reference. The rows of the matrix let you choose which of the different platforms each quality level will apply to. The Default row at the bottom of the matrix is not a quality level in itself but rather sets the default quality level used for each platform (a green checkbox in a column denotes the level currently chosen for that platform). Unity comes with six quality levels pre-enabled but own levels can also be added using the button below the matrix. The trashcan icon (the rightmost column)can be used to delete an unwanted quality level.
The name of a quality level can be selected for editing, which is done in the panel below the settings matrix:-

Name	The name that will be used to refer to this quality level
Pixel Light Count	The maximum number of pixel lights when Forward Rendering is used.
Texture Quality	This lets you choose whether to display textures at maximum resolution or at a fraction of this (lower resolution has less processing overhead). The options are Full Res, Half Res, Quarter Res and Eighth Res.
Anisotropic Textures	This enables if and how anisotropic textures will be used.

Per Texture	Anisotropic rendering will be enabled separately for each Texture.
Forced On	Anisotropic textures are always used.
AntiAliasing	This sets the level of antialiasing that will be used. The options are 2x, 4x and 8x multi-sampling.
Soft Particles	Should soft blending be used for particles?
Shadows	This determines which type of shadows should be used
Hard and Soft Shadows	Both hard and soft shadows will be rendered.
Hard Shadows Only	Only hard shadows will be rendered.
Disable Shadows	No shadows will be rendered.
Shadow resolution	Shadows can be rendered at several different resolutions: Low, Medium, High and Very High. The higher the resolution, the greater the processing overhead.
Shadow Cascades	The number of shadow cascades can be set to zero, two or four. A higher number of cascades gives better quality but at the expense of processing overhead (see the Directional Shadows page for further details).
Shadow Distance	The maximum distance from camera at which shadows will be visible. Shadows that fall beyond this distance will not be rendered.
VSync Count	Rendering can be synchronised with the refresh rate of the display device to avoid "tearing" artifacts (see below). You can choose to synchronise with every vertical blank (VBlank), every second vertical blank or not to synchronise at all.
LOD Bias	LOD levels are chosen based on the onscreen size of an object. When the size is between two LOD levels, the choice can be biased toward the less detailed or more detailed of the two models available. This is set as a fraction from 0 to 1 - the closer it is to zero, the more the bias is toward the less detailed model.
Maximum LOD Level	The highest LOD that will be used by the game. See note below for more Information.
Particle Raycast Budget	The maximum number of raycasts to use for approximate particle system collisions (those with Medium or Low quality). See Particle System Collision Module.

7.2 Build Settings

At any time while creating the game, to see how it looks after building and to run it outside of the editor as a standalone or web player, Build settings are used. File->Build Settings... is the menu item to access the Build Settings window. It pops up an editable list of the scenes that will be included when the game is build.

The Build Settings window

The first time this window is viewed in a project, it will appear blank. If the game is build while this list is blank, only the currently open scene will be included in the build. If a test player is to be quickly build with only one scene file, just build a player with a blank scene list.
It is easy to add scene files to the list for multi-scene builds. There are two ways to add them. The first way is to click the Add Current button.The currently open scene appear in the list. The second way to add scene files is to drag them from the Project View to the list.
At this point, each of the scenes have a different index value. Scene 0 is the first scene that will be loaded when the game is build. When a new scene is to be loaded, use Application. LoadLevel() inside the scripts.

If more than one scene file is added and rearrangeing them is required, simply click and drag the scenes on the list above or below others until the desired order is obtained.
If a scene is to be removed from the list, click to highlight the scene and press Command-Delete. The scene will disappear from the list and will not be included in the build.

7.3 Publishing the Game

To publish the build, select a Platform and make sure that the Unity logo is next to the platform; if it’s not, then, click in the Switch Platform button to let Unity know which platform to build from. Finally press the Build button. A name and location for the game can be selected using a standard Save dialog. On clicking Save, Unity builds the game pronto. It's that simple.

7.4 Future Development Scope

By no means, is the end of development potential for this project. In fact this is just the beginning. There are many more features that can be integrated to spice up the gameplay. These changes can definitely be brought in future versions. Some potential targets are given below.

Ports to Other Platforms	We could port this project to other platforms like Mac OS X, Android, IOS, Windows Phone etc. We need to make changes to the input methods and Poly Counts in models such that it runs efficiently on consoles.
Handling Improvements	Vehicle handling can be tweaked further to make the experience exciting. We could alter the physics options to fine tune the driveability.
Visual Improvements	There is infinite potential in thia field. High Poly Models could be introduced. Advanced Lightmapping could also result in more realistic lighting and shadows along with improved performance.
More Elements	We could introduce more car models, tracks, powerups etc. to increase the variety in choice and also adding an extra bit of fun!
Advanced Multiplayer	Multiplayer through LAN has always been a target since the beginning but we could take the experience to a new level in future releases by extending multiplayer capabilities over the Internet.
Social Integration	We could add features that could share our scores and achievments on Social Networking sites like Facebook, Twitter and Google Plus

Bibliography

Books

- **Game Development with Unity** *by Michelle Menard*
- **Unity 3.x Game Development Essentials** *by Will Goldstone*
- **Unity 3.x Game Development by Example Beginner's Guide** *by Ryan Henson Creighton*
- **Unity 3.x Scripting** *by Volodymyr Gerasimov , Devon Kraczla*
- **The Art of Maya: An Introduction to 3D Computer Graphics** *by Autodesk Maya Press*
- **The Game Artist's Guide to Maya** *by Michael McKinley*
- **Introducing Autodesk Maya 2013** *by Dariush Derakhshani*

Websites

- **http://unity3d.com/learn/documentation**
- **http://cgcookie.com/unity/**
- **http://www.unity3dstudent.com/**
- **http://www.design3.com/training-center/engines-sdks/unity/**
- **http://www.creativecrash.com/maya/tutorials/**
- **http://www.digitaltutors.com/training/maya-tutorials**

YouTube Channels

- https://www.youtube.com/user/**PolyNurb**
- https://www.youtube.com/user/**DigitalArtsGuild**
- https://www.youtube.com/user/**da1bu89**