

California State University, San Bernardino

**CSUSB ScholarWorks**

---

Theses Digitization Project

John M. Pfau Library

---

2011

## An open source infrastructure for 3D virtual worlds

Amita Kale

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Software Engineering Commons](#)

---

### Recommended Citation

Kale, Amita, "An open source infrastructure for 3D virtual worlds" (2011). *Theses Digitization Project*. 3924.

<https://scholarworks.lib.csusb.edu/etd-project/3924>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact [scholarworks@csusb.edu](mailto:scholarworks@csusb.edu).

AN OPEN SOURCE INFRASTRUCTURE  
FOR 3D VIRTUAL WORLDS

---

A Project  
Presented to the  
Faculty of  
California State University,  
San Bernardino

---

In Partial Fulfillment  
of the Requirements for the Degree  
Master of Science  
in  
Computer Science

---

by  
Amita Kale  
June 2011

AN OPEN SOURCE INFRASTRUCTURE  
FOR 3D VIRTUAL WORLDS

---

A Project  
Presented to the  
Faculty of  
California State University,  
San Bernardino


---

by

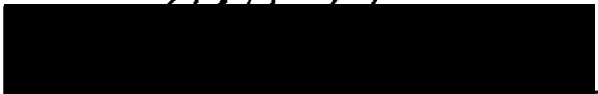
Amita Kale

June 2011

Approved by:

  
Dr. David A. Turner, Advisor, School of  
Computer Science and Engineering

6/8/2011  
Date

  
Dr. Arturo I. Concepcion

  
Dr. Tong Lai Yu

© 2011 Amita Kale

## ABSTRACT

Geng is an ongoing project at the School of Computer Science and Engineering at California State University San Bernardino. Geng is derived from the words “Game” and “Engine”. Geng is a platform independent library that can be used to construct 3D virtual worlds. The Geng library provides an API that can be used by applications to create 3D games or simulations. The Geng library is written to facilitate integration with application code written in other languages such as C++, Java, C#, Python, etc. In addition to 3D graphics, the Geng library also provides integration with the Bullet Physics Library through the Geng Physics API. Geng has gone through three iterations in past. The first iteration included rendering models and integration with the Bullet Physics Library. The second iteration included implementing game AI using expert systems. The third iteration included creating a multi platform graphics engine in which, CG shaders were used to render the game assets and Simple Direct Media Layer(SDL) library was used to handle input events. This project is the fourth iteration of Geng. The main purpose of this iteration is to re-structure and enhance the Geng library. The re-structuring involves defining and adhering to clearly stated organizational principles and coding standards. The enhancements involve eliminating the dependencies on third party applications Simple Direct Media Layer(SDL) and CG. It also involves creating exporters for the 3D modeling applications Maya and Blender.

## ACKNOWLEDGEMENTS

I would like to thank my project advisor Dr. David Turner for his guidance, help, and support. I would also like to thank my committee members Dr. Arturo I. Concepcion and Dr. Tong Lai Yu for their help and support. I would like to thank Dr. Mendoza, my graduate advisor.

Last but not the least I would like to thank my husband Chinmay for his love and support.

## TABLE OF CONTENTS

<i>Abstract</i> . . . . .	iii
<i>Acknowledgements</i> . . . . .	iv
<i>List of Tables</i> . . . . .	viii
<i>List of Figures</i> . . . . .	ix
<b>1. Introduction</b> . . . . .	1
1.1 Background . . . . .	1
1.2 Purpose . . . . .	1
1.3 Project Scope . . . . .	2
1.4 Project Limitations . . . . .	2
<b>2. System Architecture and Organizational Principles</b> . . . . .	4
2.1 Restructuring . . . . .	4
2.2 Geng API . . . . .	6
2.3 Organizing Principles . . . . .	9
2.3.1 Restrict global functions to the Geng API. . . . .	9
2.3.2 Associate handles with pointers through C++ maps. . . . .	10
2.3.3 Follow capitalization rules. . . . .	11
2.3.4 Use consistent singleton patterns. . . . .	11
2.3.5 Put platform dependent logic/code in sub-classes. . . . .	12

2.3.6	Assets should be in the Geng specific format. . . . .	13
2.4	Coding Standards . . . . .	14
3.	<i>Graphical User Interface (GUI)</i> . . . . .	15
3.1	Geng GUI . . . . .	15
3.2	Geng GUI Primitive . . . . .	17
4.	<i>Scene Rendering</i> . . . . .	25
4.1	Geng Scene . . . . .	25
4.2	Geng Scene Primitive . . . . .	26
5.	<i>Picking</i> . . . . .	34
5.1	Object Picking . . . . .	34
5.2	Picking in Geng . . . . .	36
6.	<i>Art Asset Processing</i> . . . . .	40
6.1	Shaders for Asset Rendering . . . . .	40
6.2	Geng Asset Conversions . . . . .	42
6.3	Image Converter . . . . .	44
6.4	Exporter for Maya . . . . .	45
6.5	Exporter for Blender . . . . .	50
7.	<i>Geng Library Usage</i> . . . . .	57
7.1	Geng Initializations . . . . .	57
7.2	Geng Graphics . . . . .	58
7.3	Event Handling . . . . .	59
7.4	Geng Physics Library . . . . .	61
8.	<i>Conclusion and Future Direction</i> . . . . .	65
8.1	Conclusion . . . . .	65



8.2 Future Direction . . . . .	66
8.3 Definitions, Acronyms, and Abbreviations . . . . .	67
<i>References</i> . . . . .	70

## LIST OF TABLES

2.1	Dynamic Library Files Produced from the Geng Project . . . . .	7
5.1	Pickable Object Map . . . . .	37

## LIST OF FIGURES

2.1	The Previous Geng Architecture . . . . .	5
2.2	The New Geng Architecture . . . . .	6
2.3	The Four Builds of Geng . . . . .	8
2.4	Graphics Class Hierarchy . . . . .	13
3.1	Screen Positions . . . . .	17
3.2	The GuiPrimitive Class and its Associated Classes . . . . .	19
3.3	The GuiVertex Class . . . . .	20
3.4	The VertexBuffer Class . . . . .	20
3.5	The IndexBuffer Class . . . . .	21
3.6	The Shader Class . . . . .	21
3.7	The Matrix Class . . . . .	22
3.8	The Texture Class . . . . .	22
3.9	The Font Class . . . . .	22
3.10	Rendering GUI Primitives . . . . .	24
4.1	The ScenePrimitive Class and its Associated Classes . . . . .	28
4.2	The ModelVertex Class . . . . .	29
4.3	The VertexBuffer Class . . . . .	29
4.4	The IndexBuffer Class . . . . .	30
4.5	The Shader Class . . . . .	30
4.6	The Matrix Class . . . . .	31

4.7	The Texture Class . . . . .	31
4.8	Rendering Scene Primitives . . . . .	33
5.1	The Picker Class . . . . .	35
5.2	Normal Rendering . . . . .	38
5.3	Pick Rendering . . . . .	39
6.1	Geng_image File Format . . . . .	42
6.2	Geng_model File Format . . . . .	43
6.3	Maya Application . . . . .	47
6.4	Maya Application Creating a Mesh . . . . .	48
6.5	Select Texture File . . . . .	49
6.6	Maya Application Texturing a Mesh . . . . .	50
6.7	Blender Application . . . . .	53
6.8	Mesh Menu . . . . .	54
6.9	UV Image Editor . . . . .	55
6.10	Load Texture . . . . .	55
6.11	Draw Type Textured . . . . .	56
6.12	Export . . . . .	56
7.1	Entity Class Hierarchy . . . . .	63

## 1. INTRODUCTION

### *1.1 Background*

Geng is an ongoing project at the School of Computer Science and Engineering at California State University San Bernardino (CSUSB). Geng is derived from the words “Game” and “Engine”. Geng has gone through three major iterations in the past. The first iteration was done by William Herrera, a former CSUSB student. His contribution included terrain generation, integration with the Bullet Physics Library [2], loading and rendering of models, and implementing a multi-player system. The second iteration was done by Christopher Ballinger, a former CSUSB student. He experimented with embedding a web browser inside the game. He also implemented game AI using expert system called Drools [8]. The third iteration was done by David Stover, a former CSUSB student. He created a Multi-Platform All-Purpose Graphics Engine. He integrated Geng with the Bullet Physics Library. He used CG shaders to render 3D models and animations [11].

### *1.2 Purpose*

This project is the fourth iteration of Geng. The purpose of this iteration is three-fold. The first purpose is to completely re-structure the Geng library, to improve the efficiency and the maintainability of the code. The second purpose is to enhance the Geng library by eliminating the dependencies on third party applications and adding utilities to broaden the scope of assets, which can be rendered using the Geng library.

The third purpose is to build exporters to export 3D models from 3D modeling applications to the Geng specific file format.

### *1.3 Project Scope*

The scope of this project includes creating a new architecture and defining organizational principles for the Geng library. The Geng library provides a set of functionality in the form of an API. The functions in the API can be used to create the following.

- Scientific simulations
- Multi-player games
- Shared virtual worlds

This project also eliminates dependencies on the third party applications Simple Direct Media Layer(SDL) and CG shaders. Simple Direct Media Layer(SDL) was used in the previous Geng library, mainly to create the application window and to handle input events. CG shaders were used by the previous Geng library to render assets. However, in this project, these dependencies are eliminated. In place of Simple Direct Media Layer(SDL), this project uses the operating system functions, to create the application window and to handle input events. Also, in place of CG shaders, this project uses OpenGL Shader Language(GLSL) and High Level Shader Language(HLSL) to program the video system. This project provides two exporters to convert assets created in the 3D modeling applications Maya [7] and Blender [1] to the Geng specific file format.

### *1.4 Project Limitations*

This project has the following limitations.

- The Geng library can not be used as a complete general purpose game engine to create a game; it provides a basic starting point for a 3D application. However, eventually the users will have to make changes to the Geng library to match their requirements.
- The Geng library currently does not support animated 3D models and particle systems.

## 2. SYSTEM ARCHITECTURE AND ORGANIZATIONAL PRINCIPLES

### *2.1 Restructuring*

Re-structuring of the previous Geng library is motivated by its architectural limitations. The previous Geng architecture is shown in Fig. 2.1. This diagram was taken from [11]. It consists of components such as core graphics, GUI, text, video and model. Core graphics include the functionality that uses the OpenGL and DirectX components.



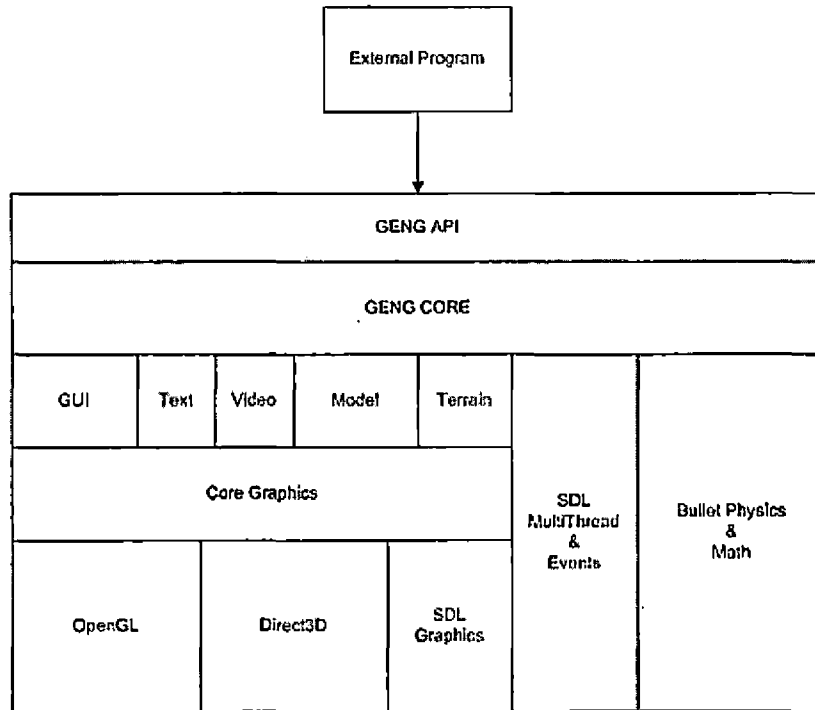


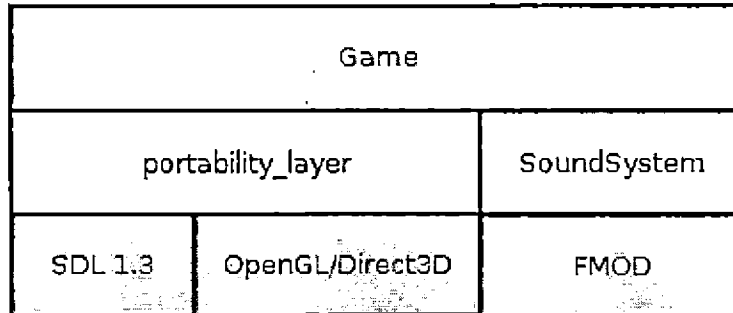
Fig. 2.1: The Previous Geng Architecture

The limitations of the previous architecture are as follows.

- The separation of concerns shown in Fig. 2.1 is not clearly expressed in the code. The existing code can be re-organized into separate libraries to increase maintainability of the code.
- There is a dependency on the third party applications Simple Direct Media Layer(SDL) and CG shaders. Simple Direct Media Layer(SDL) is used to create an application window and to handle input events, whereas CG shaders are used to render assets. This adds complexity to the build and confines Geng to the limitations of these libraries.
- There are inadequate coding conventions and organization principles followed, which makes it difficult to maintain and enhance the code.

- Previously only .x file format was supported for loading static 3D models.

The new Geng architecture is an attempt to overcome all the limitations listed above. Fig. 2.2 shows the new Geng architecture.



*Fig. 2.2: The New Geng Architecture*

The new Geng architecture has a clearly defined API. This API acts as a platform independent graphics engine providing graphics functionality to the user application. It internally accesses OpenGL or DirectX graphics APIs depending on the platform. Sound System is a C# wrapper for FMOD [4] functions. This hides the complexity of FMOD functions from the user application. Sound System functions can be used by the user application to get interactive audio functionality. Geng Physics is a C++ wrapper for the Bullet Physics Library. Bullet is an open source physics library that provides the functionality such as soft and rigid body dynamics and collision detection. Geng Game is our example application that uses functions from the Geng API. Geng Game is written in C#. However, Geng Game can be written in C++, Python, Java or any other language that integrates with libraries that have C linkage.

## 2.2 Geng API

Geng is a library that provides a set of functionality to facilitate creation of real-time graphical simulations such as video games and scientific simulations. The Geng

library is implemented for the common desktop operating systems Linux, Windows and OSX. For Windows, there are two versions of the library: one that relies on OpenGL and another that relies on DirectX. Therefore, there are four builds of the Geng project, which lead to four distributable binary versions of the library. These library binaries are listed in Tab. 2.1

Dynamic Libraries	Description
geng_win_dx.dll	Geng API for Windows based applications that make use of DirectX API for graphics.
geng_win_gl.dll	Geng API for Windows based applications that make use of OpenGL API for graphics.
geng_osx.dylib	API for Mac based applications that make use of OpenGL API for graphics.
geng_linux.so	Geng API for Linux based applications that make use of OpenGL API for graphics.

Tab. 2.1: Dynamic Library Files Produced from the Geng Project

The Geng library internally uses object-orientated techniques. However, its exported functions have C linkage. The reason for this is to make use of object-oriented architecture in order to improve readability of the code, yet provide a C-style interface to outside users of the code in order to make it easy to integrate Geng with code written in other languages. Another reason to give the exported functions C linkage is to provide a standard object format that can be used by all compilers. (C++ linkage is not standardized, and not all compilers produce compatible library files.) The Geng library has been tested with programs written in C++, C# and Java.

The exported functions in Geng are referred to as the Geng API. This interface does not include classes, enums, strings, and other C++ constructs that do not appear in the C language because it is designed as a C interface. However, C++ classes, enums,

and strings are used extensively in the internal implementation of the Geng API.

One of the architectural principles that was followed was to minimize the use of global C-style functions. However, global C-style functions are needed to implement the API functions, and they were also convenient in a few cases for implementing callback functions needed to interact with the operating system and other libraries. The implementation of the Geng API functions was organized into seven files. This was done to allow maximum reuse of identical code across the different builds of the Geng library. These seven API implementation files are shown in Fig. 2.3.

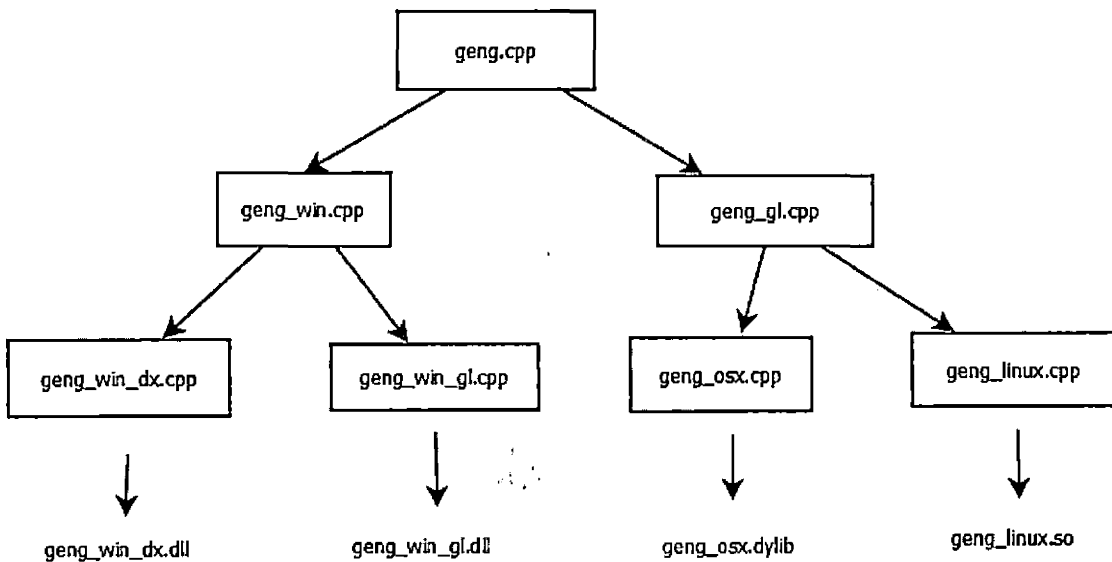


Fig. 2.3: The Four Builds of Geng

All the seven files shown in the Fig. 2.3 directly implement the functions declared in the Geng API. The file `geng.cpp` contains the code that works for all the builds. This includes functions such as `Tick()`, which is used to process user input events and generate video frames. The file `geng_win.cpp` contains the code that works for the two Windows builds. This code is responsible for creating an application window, handling windows events, and initializing the graphics sub-system. The file `geng_win_dx.cpp` contains the code that works for the Windows build for DirectX. The

file `geng_win_gl.cpp` contains code that works for the Windows build for OpenGL. The file `geng_gl.cpp` contains the code that works for OpenGL builds for Linux and OSX. The file `geng_osx.cpp` contains the code that works for the OSX build for OpenGL. The file `geng_linux.cpp` contains the code that works for the Linux build for OpenGL.

## 2.3 Organizing Principles

The Geng library adheres to the organizational principles that are listed as follows.

1. Restrict global functions to the Geng API.
2. Associate handles with pointers through C++ maps.
3. Follow capitalization rules.
4. Use consistent singleton patterns.
5. Put platform dependent logic/code in sub-classes.
6. Assets should be in the Geng specific format.

Each of the organizational principles listed above are explained in detail as follows.

### 2.3.1 *Restrict global functions to the Geng API.*

All the global functions are restricted only to the Geng API. The Geng API functions are global, C functions. The code that directly implements these API functions calls into the internal object oriented architecture. Thus, all global functions should be a part of the Geng API. All the other functionality should be written in the form of classes and objects.

Consider the function `SetBackgroundColor()`. This function sets the background color of the application window according to the color values passed to it. The function is declared in `geng.h` as a global function as shown in the following code.

```
GENG_API void SetBackgroundColor(float red, float green, float blue, float alpha);
```

However the implementation of this function is as follows.

```
void SetBackgroundColor(float red, float green, float blue, float alpha)
{
    graphics->setBackgroundColor(red, green, blue, alpha);
}
```

In the above code, the `setBackgroundColor()` function of the `Graphics` class is invoked through the `graphics` pointer.

### *2.3.2 Associate handles with pointers through C++ maps.*

The user application can create Geng primitives, such as GUI primitive objects or scene primitive objects using the functions provided by the Geng API. To perform operations on a primitive object, the user application has to pass a reference of that particular primitive object to the Geng API. However, as mentioned earlier, the Geng API has C linkage. Hence, pointers to the primitive objects cannot be directly passed through the Geng API. To resolve this problem, the Geng API uses integer handles that correspond to the Geng primitive objects. When a Geng primitive is created by the user application using a Geng API function, the Geng API function returns an integer handle corresponding to that Geng primitive object. This handle is stored by the user application and can be used to perform operations on the Geng primitive. Inside Geng, this integer handle is converted into an object pointer using a C++ map that associates the integer handles with the object pointers.

The following example shows how the maps associating handles with pointers work.

To use Geng to implement a label, the user application needs to call two functions from the Geng API. Firstly, the user application has to call `CreateGuiPrimitive()` in order to create an instance of a GUI primitive. This call is shown in the following code sample.

```
int titleHandle = Geng.CreateGuiPrimitive
    (Geng.ScreenPositionUpperCenter ,
     0, -40, Geng.True);
```

As shown in the code sample above, `CreateGuiPrimitive()` function returns the integer handle corresponding to the GUI primitive. The user application stores this handle. Also internally Geng creates a map entry for the GUI primitive handle corresponding to the GUI primitive object pointer.

Secondly, the user application should call `SetGuiPrimitiveLabel()` from the Geng API to configure the GUI primitive to render a string of text. To do this we need to pass the GUI primitive handle as well as the font to use and the string to render. This function invocation is shown in the following code.

```
Geng.SetGuiPrimitiveLabel(titleHandle, "fonts/miso_32", 0, "Welcome", 0, 0);
```

The `SetGuiPrimitiveLabel` function converts the integer handle passed by the user application to GUI primitive object pointer.

### *2.3.3 Follow capitalization rules.*

The following capitalization rules should be followed by the Geng library.

- All the API functions should start with the first letter capitalized. This is to be consistent with C#, which is the primary user language of the Geng API.
- All the .h files that have class declarations should be first letter capitalized.
- All the classes should start with the first letter capitalized.
- All the class member functions should start with the first letter lower cased.

### *2.3.4 Use consistent singleton patterns.*

All the platform independent code in the Geng library is coded in the generic parent classes and the platform dependent code is coded in a sub-class of the platform

independent class. These platform independent classes are singletons and a specific style is used to access the platform dependent code using these singletons.

Consider the following example.

Graphics is a generic class that has sub-classes GraphicsDx and GraphicsGl as shown in Fig. 2.4. These sub-classes contain the platform specific code. Graphics is a singleton class. The following pattern is used to instantiate it and access the platform dependent code.

In GraphicsDx.cpp and GraphicsGl.cpp, a pointer to the sub-class is created and it is assigned to the pointer to the base class as shown in the following code.

For Dx:

```
GraphicsDx * graphicsDx = new GraphicsDx();
```

```
Graphics * graphics = graphicsDx;
```

For Gl:

```
GraphicsGl * graphicsGl = new GraphicsGl();
```

```
Graphics * graphics = graphicsGl;
```

Wherever there is need to access the platform dependent code from the Graphics class, the pointer is declared as shown in the following code.

```
extern Graphics *graphics;
```

This graphics pointer can be used to access the member functions of the Graphics class which in turn would call the platform specific functions.

### 2.3.5 Put platform dependent logic/code in sub-classes.

The object oriented code in the Geng library is divided into two types of classes: the generic parent classes, which have the functionality common across the platforms and the sub-classes, which define platform specific code. The generic parent classes have pure virtual functions. These functions are defined in the sub-classes. Consider the Graphics class. Graphics is a generic parent class that has the functionality



common across the platform. The Graphics class also has pure virtual functions, which have different definitions in the platform specific sub-classes: GraphicsDx and GraphicsGl. Consider the init() function. It is a pure virtual function that is declared in the Graphics class and is defined in GraphicsDx and GraphicsGL respectively. The following Fig. 2.4 shows the Graphics class hierarchy.

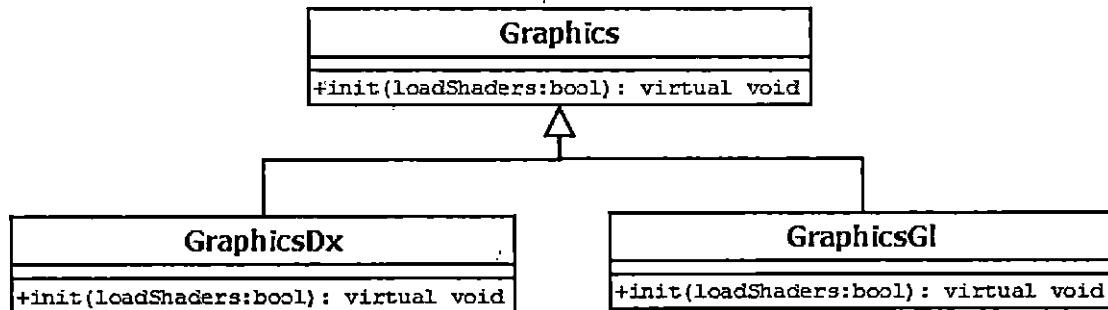


Fig. 2.4: Graphics Class Hierarchy

The init() function is declared as pure virtual and defined in the sub-classes.

### 2.3.6 Assets should be in the Geng specific format.

Assets are the building blocks of any game or simulation. Assets can be images or static models that need to be rendered in the game. Geng has different types of assets such as images, static 3D models, font descriptors etc.

The format of these assets should be converted into the format that is specific to Geng. The Geng specific format makes it simpler to run platform independent code. Thus, all the assets must be converted into the Geng specific format before loading them in Geng. This is done using the exporters. Two exporters are written for the 3D modeling applications, Maya and Blender. These exporters convert static models created using Maya and Blender into Geng specific file format. The model file names end with .geng\_model. Also there is an asset conversion tool to convert images to the Geng specific file format. The Geng image file names end with geng\_image format.

Having Geng specific formats for the assets makes it easier to render them in any platform. In addition, it moves the code complexity to the exporters, keeping the implementation of the Geng API simpler.

## 2.4 Coding Standards

The advantages of specifying and adhering to a coding standard include the following.

- Enforcing the style increases the readability of code.
- Coding standards make the code consistent, which increases maintainability of code.
- The coding standards allow the use of C++ language features more effectively.

Geng follows the Google style guide for C++ [6] coding with several exceptions. These exceptions are as follows.

- Variables and constants naming conventions are different for Geng. They start with lower case and have no underscores. Consider the following sample code that shows how variables are declared in the Geng library.

```
static int sceneViewHandle = 0;
static int nextHandle = 1;    // Zero handle means null pointer.
```

- Functions naming rules are different for Geng. All the functions start with the lower case except for the Geng API functions, which start with the upper case. Consider the following sample code that shows how the Geng API functions are declared in the Geng library.

```
GENG_API void InitWindowed(unsigned int screenWidth,
                           unsigned int screenHeight,
                           const char * title);
GENG_API void SetWindowClosingHandler(WindowClosingHandler windowClosingHandler);
```

### 3. GRAPHICAL USER INTERFACE (GUI)

#### 3.1 *Geng GUI*

The graphical user interface (GUI) is an important part of any graphical application. For the gaming application or any graphical simulation, GUI plays an important role to display status information, provide navigation etc. For a game application, GUI can also be referred as the Heads Up Display (HUD). The HUD is used to display player related information such as the game level, player's health, weapons etc.

The Geng library provides a single GUI primitive object to the users of this library. This GUI primitive can be used to render a line of text or an image. This can be used in combination to create controls such as labels, input boxes, buttons, images and other controls.

The GUI is rendered on the top of a 3D scene. When the user clicks anywhere on the application window, first it is checked if a GUI primitive is clicked. Then it is checked if a scene primitive is clicked. Thus the GUI primitive has the highest priority for the mouse click event.

In the Geng library, the GUI class functions as a manager of the GUI primitives. This class maintains the list of all the GUI primitives created by the user application. The GUI class makes use of the following functions to manage the GUI primitives.

- void init()

This function initializes the GUI primitive, creates an empty quad and initializes an orthographic projection matrix.

- `void determineScreenCoordinates(int screenPosition, int xOffset, int yOffset, int * screenX, int * screenY)`  
This function returns the screen co-ordinates, which are screenX and screenY depending on the given screen position and offset.
- `Matrix * getGuiOrthoProjectionMatrix()`  
This function returns a pointer to an orthographic projection matrix.
- `IndexBuffer * getQuadIndexBuffer()`  
This function returns a pointer to the index buffer used for rendering GUI primitives.
- `void addGuiPrimitive(GuiPrimitive * guiPrimitive)`  
This function adds the GUI primitive to the list of the GUI primitives that are rendered. This is done when the user code sets the GUI primitive visibility to true.
- `void removeGuiPrimitive(GuiPrimitive * guiPrimitive)`  
This function removes the GUI primitive from the list of the GUI primitives being rendered. This is done when the user code sets the GUI primitive visibility to false.
- `void render(bool picking)`  
This function renders all the GUI primitives in the list of the GUI primitives to be rendered. If the picking parameter is set to true, then the pick shader is activated and pick rendering mode is selected. If the picking parameter is set to false, then the normal rendering happens.
- `void removeAllGuiPrimitives()`  
This function removes all the GUI primitives from the list of the GUI primitives to be rendered.

### 3.2 Geng GUI Primitive

The Geng library provides the functions to create a GUI primitive. Creating a GUI primitive is a two step process.

The first step involves creating an empty GUI primitive and the second step involves setting the GUI primitive to display text or display image or display both. The following is the code for the function that is used to create a GUI primitive. This function returns the integer handle to that GUI primitive, which is used for invoking subsequent operations on the GUI primitive.

```
GENG_API int CreateGuiPrimitive(  
    int screenPosition,  
    int xOffset,  
    int yOffset,  
    int visible);
```

In Geng, the application screen is divided into nine regions as shown in Fig. 3.1.

Upper Left	Upper Center	Upper Right
Middle Left	Dead Center	Middle Right
Lower Left	Lower Center	Lower Right

Fig. 3.1: Screen Positions

The screen position value identifies on where on the screen the user wants the GUI primitive to be rendered. The `xOffset` and the `yOffset` are the x, y offsets from

the origin of the given screen position for the GUI primitive. The visible flag sets the visibility for the GUI primitive. When its value is true, then the GUI primitive is visible. The `CreateGuiPrimitive()` function returns an integer handle to the GUI primitive.

Once the handle to the GUI primitive is acquired, then according to the user's requirement, the GUI primitive can be set to display either image or text or both. The following code sample shows the function that is used to set the GUI primitive to render an image.

```
GENG_API void SetGuiPrimitiveImage(  
    int guiPrimitive,  
    const char * gengImageFilename);
```

This function tells the GUI primitive to render an image. The integer handle of the GUI primitive is passed to the `SetGuiPrimitiveImage()` function along with the image filename for the image that needs to be rendered.

The following is a function that is used to set the GUI primitive to render text.

```
GENG_API void SetGuiPrimitiveLabel(  
    int guiPrimitive,  
    const char * fontName,  
    int maxCharacters,  
    const char * text,  
    int textOffsetX,  
    int textOffsetY);
```

This function tells the GUI primitive to render text. The integer handle is passed to the `SetGuiPrimitiveLabel()` function along with the font and the text that needs to be rendered. The following is the C# code sample that uses the Geng API functions to render the GUI primitive text.

```
int titleHandle =
```

```

Geng.CreateGuiPrimitive(Geng.ScreenPositionUpperCenter,
    0, -40, Geng.True);
Geng.SetGuiPrimitiveLabel(
    titleHandle,
    "fonts/miso_32",
    0, "Welcome", 0, 0);

```

The code above, renders the GUI text “Welcome”. CreateGuiPrimitive() returns the handle to the GUI primitive. This handle is passed to the SetGuiPrimitiveLabel() along with the font and the string to be rendered.

The Geng library for now supports only one font miso\_32. The font files are generated using AngleCode Bitmap Font Generator. It runs under windows and generates bit-mapped fonts from true type fonts. The files miso\_32.fnt and miso\_32.tga were generated from the AngleCode Bitmap Font Generator. The miso\_32.fnt file was converted into miso\_32.geng\_image file to transform it into the Geng specific image file format. The image file has the actual letters and numbers for the font and the miso\_32.fnt file has information such as width, height, kerning etc.

The GuiPrimitive class is the main class that has functions to create the GUI primitives. The GuiPrimitive class has the associated classes as shown in the Fig. 3.2.

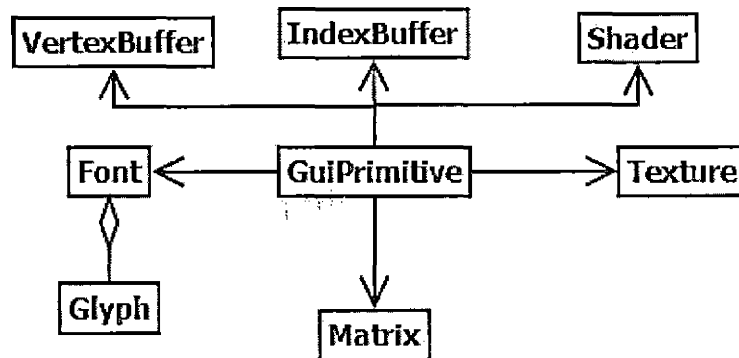


Fig. 3.2: The GuiPrimitive Class and its Associated Classes

All the classes in the Fig. 3.2 are explained as follows.

The VertexBuffer class creates a vertex buffer. A vertex buffer contains the vertex data. The GuiVertex class defines the data each vertex holds for a GUI primitive. As shown in Fig. 3.3, the GuiVertex class holds x, y, z co-ordinates and u,v texture co-ordinates.

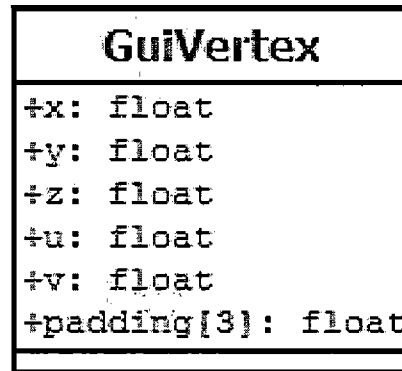


Fig. 3.3: The GuiVertex Class

A vertex buffer is implemented differently in OpenGL and DirectX. Hence, there are different classes for the OpenGL and DirectX implementations as shown in Fig. 3.4.

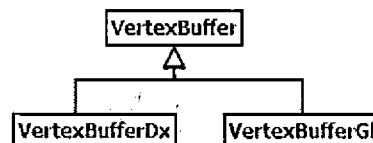


Fig. 3.4: The VertexBuffer Class

The IndexBuffer class creates an index buffer. An index buffer contains the list of vertices in the required drawable order. An index buffer is implemented differently in OpenGL and DirectX. So there are different classes for the OpenGL and DirectX implementations as shown in Fig. 3.5.



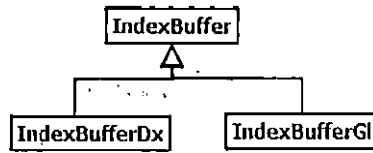


Fig. 3.5: The IndexBuffer Class

The GUI primitives are rendered using shaders. The Shader class gives access to the shader programs. The Shader class functions activate shaders and pass parameters to them. There are two types of shader languages, High Level Shader Language(HLSL) and OpenGL Shader Language(GLSL).

HLSL shader code for the GUI primitives is stored in a file called gui.fx. This gui.fx file has the code for both vertex shaders and fragment shaders.

GLSL shader code for the GUI primitives is stored in two files gui.vert and gui.frag. The gui.vert file has the code for vertex shader and the gui.frag file has the code for the fragment shaders.

The shaders are implemented differently for OpenGL and DirectX. Hence there are different implementations of the Shader class for the different platforms. This is shown in Fig. 3.6.

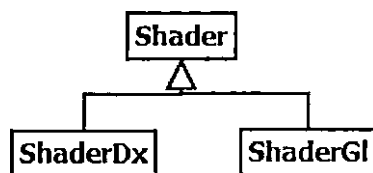


Fig. 3.6: The Shader Class

The Matrix class is used to set the world matrix and projection matrix. Matrices are set differently in DirectX and OpenGL so there are different implementations of the Matrix class as shown in Fig. 3.7.

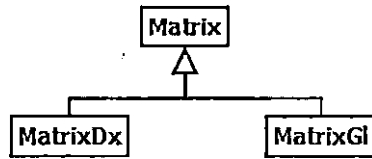


Fig. 3.7: The Matrix Class

The Texture class is used to load images for the GUI primitives. It gets the x, y and u,v co-ordinates for the image that are used to render GUI primitive Image. Textures are implemented using different derived classes for DirectX and OpenGL as shown in Fig. 3.8.

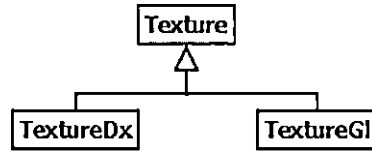


Fig. 3.8: The Texture Class

The Font class is used by the GUI primitive to render text. To render a single character, a quad is created and is textured with the image from the font image file. The information about the position of a character in the image file and also the height and the width of character is stored in the font descriptor file and is retrieved using the Glyph structure.



Fig. 3.9: The Font Class

The Geng Library provides the functionality to handle events for the GUI primitives. The GUI primitives such as text and image can be associated with events such as mouse clicks. To check if any GUI primitive is clicked, a picking technique is used.

This picking technique is explained in detail in chapter 5. The following C# code sample shows how to handle mouse click events.

```
int woodHandle = Geng.CreateGuiPrimitive(  
Geng.ScreenPositionDeadCenter, 0, 0, Geng.True);  
Geng.SetGuiPrimitiveImage(woodHandle, "gui/wood_64_64.geng_image");  
Geng.SetGuiPrimitiveHandler(woodHandle, HandleTitleClick);  
public void HandleTitleClick(int handle)  
{  
    Geng.SetKeyHandler(null);  
    Geng.UnloadGraphics();  
    Program.Instance.ChangeState(null);  
}
```

After creating an empty GUI primitive and setting it to render an image, `SetGuiPrimitiveHandler()` function is called. This function needs to be called to associate an integer event handle with a function. Here the function `HandleTitleClick()` is associated with the handle for the GUI primitive. Thus the function `HandleTitleClick()` is called on the mouse click event. The images that are used to texture the GUI primitives can be created using image manipulation programs such as GIMP. These images have to be converted to the Geng specific file format using asset conversion tool or using the Geng exporters for Blender or Maya.

The following Fig. 3.10 shows how GUI primitives are rendered on the application window. It shows three GUI primitives. One is rendered as text which is "Geng Game" and the other two are rendered as images.



*Fig. 3.10: Rendering GUI Primitives*

## 4. SCENE RENDERING

### 4.1 *Geng Scene*

The scene is an important part of any game or graphical simulation. It is the 3D space in which game objects move around or are placed in. In a typical 3D game, a scene consists of static 3D objects such as trees and houses, game characters (which can be static or animated 3D models), and particle systems such as clouds and water.

In Geng, the scene is rendered using the scene primitives. These scene primitives can be static models, animated models or particle systems. Currently Geng supports rendering of only static models.

While rendering the application, the scene primitives are rendered before the GUI primitives. So the GUI primitives are rendered on the top of scene primitives.

The difference between rendering the GUI and rendering the scene is that the GUI is rendered in 2D with no depth value. However scene is 3D space, which is rendered with the Z-buffer enabled. The Z-buffer is also called the depth buffer, which stores the depth information. The Z-buffer allows rendering of geometry behind other geometry. While rendering GUI objects the user needs to provide x and y co-ordinates. However, while rendering scene objects, the user needs to provide x, y, and z co-ordinates. The z co-ordinate represents the depth.

In the Geng library, scene class functions as a manager of the scene primitives. This class maintains the list of all the scene primitives created by the user application. The Scene class makes use of the following functions to manage the scene primitives.

- `void addScenePrimitive(ScenePrimitive * scenePrimitive)`

This function adds the scene primitive to the list of the scene primitives that are rendered. This is done when the user code sets the scene primitive visibility to true.

- `void removeScenePrimitive(ScenePrimitive * scenePrimitive)`

This function removes the scene primitive from the list of the scene primitives being rendered. This is done when the user code sets the scene primitive visibility to false.

- `void render(bool picking)`

This function renders all the scene primitives in the list of the scene primitives to be rendered. If the picking parameter is set to true, then the pick shader is activated and pick rendering mode is selected. If the picking parameter is set to false, then the normal rendering happens.

- `void init()`

This function initializes the scene primitive, sets the view matrix and sets projection matrix.

- `void removeAllScenePrimitives()`

This function removes all the scene primitives from the list of the scene primitives to be rendered.

- `Matrix * getViewMatrix()`

This function returns the pointer to the view matrix.

## 4.2 Geng Scene Primitive

The process of creating a scene primitive is a two step process

The first step involves creating an empty scene primitive and the second step involves setting the scene primitive to display a 3D static model. The following is the code for the function that is used to create a scene primitive. This function returns the integer handle to that scene primitive, which is used for invoking subsequent operations on the scene primitive.

```
GENG_API int CreateScenePrimitive(  
    float axisX, float axisY, float axisZ,  
    float angle,  
    float x, float y, float z,  
    int visible);
```

The parameters axisX, axisY, axisZ, angle, x,y,z decide the scene primitives position in the screen space. The value for axisX is set to 1 if there is a rotation around x axis. The value for axisY is set to 1 if there is a rotation around y axis. The value for axisZ is set to 1 if there is a rotation around z axis. The angle is the angle of rotation. x, y and z are displacements along x-axis, y-axis and z-axis respectively. The visible flag sets the visibility for the scene primitive. This function returns the handle to the scene primitive. This handle is stored by the user application.

The following code shows the function provided by Geng API to set the scene primitive to render a 3D model.

```
GENG_API void SetScenePrimitiveModel(  
    int scenePrimitive,  
    const char * modelFilename);
```

This function tells the scene primitive to render a static 3D model. It takes as an argument, an integer handle to the scene primitive and the model file name and displays a 3D static model. The following is a C# code sample, which shows how to set the scene primitive to render a 3D model.

```
int scenePrimitiveHandle =
```

```

Geng.CreateScenePrimitive(
    0, 0, 0, 0, -10, 0, -1, 1);
Geng.SetScenePrimitiveModel(
    scenePrimitiveHandle,
    "scene/tiger.geng_model");

```

In the example above, a scene primitive is created and a model is assigned to the scene primitive. Tiger.geng\_model is the static 3D model in geng compatible format.

The ScenePrimitive class is the main class that has functions to create the scene primitives. The ScenePrimitive class has the associated classes as shown in the Fig. 4.1.

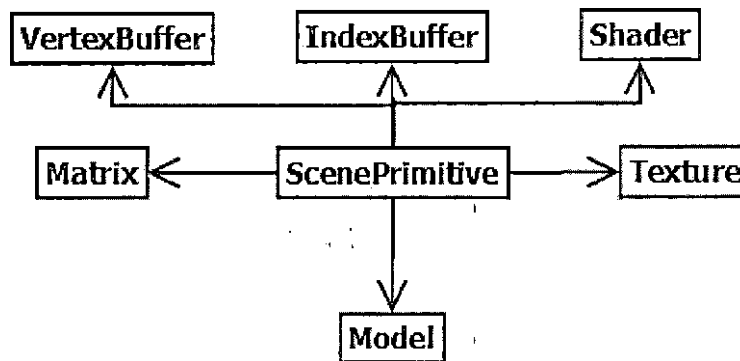


Fig. 4.1: The ScenePrimitive Class and its Associated Classes

All the classes in the Fig. 4.1 are explained as follows.

The VertexBuffer class creates a vertex buffer. A vertex buffer contains the vertex data for the scene primitive. The ModelVertex class defines the data each vertex holds for a scene primitive. As shown in Fig. 4.2, the ModelVertex class holds x, y, z co-ordinates, normals nx, ny, nz and u,v texture co-ordinates.



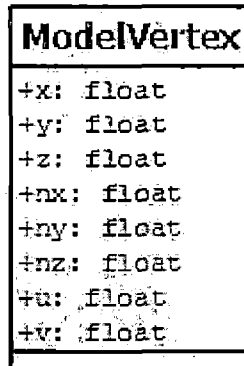


Fig. 4.2: The ModelVertex Class

A vertex buffer is implemented differently in OpenGL and DirectX. So there are different classes for the OpenGL and DirectX implementations as shown in the following Fig. 4.3.

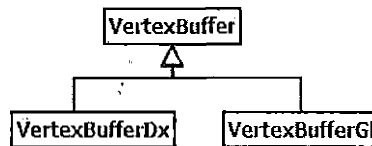


Fig. 4.3: The VertexBuffer Class

In VertexBufferGl class, the information about the location of vertex data(x, y, z), texture co-ordinates(u, v) and normals(nx, ny, nz) in the ModelVertex is defined using the sample code shown as follows.

```

//point to the normals.
glNormalPointer(GL_FLOAT, sizeof(ModelVertex), BUFFER_OFFSET(normalDataOffset));
//point to the texture co-ordinates.
glTexCoordPointer(2, GL_FLOAT, sizeof(ModelVertex), BUFFER_OFFSET(uvDataOffset));
//point to the vertex data.
glVertexPointer(3, GL_FLOAT, sizeof(ModelVertex), BUFFER_OFFSET(0));
  
```

The IndexBuffer class creates an index buffer. An index buffer contains the list of

vertices in the required drawable order. An index buffer is implemented differently in OpenGL and DirectX. Hence, there are different classes for the OpenGL and DirectX implementations as shown in the following Fig. 4.4.

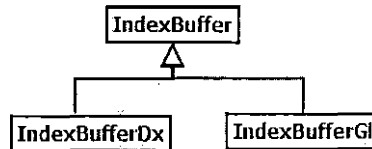


Fig. 4.4: The IndexBuffer Class

The scene primitives are rendered using shaders. The Shader class gives the access to the shader programs. The Shader class functions activate shaders, pass parameters to them. There are two types of shader languages, High Level Shader Language(HLSL) and OpenGL Shader Language(GLSL)..

HLSL shader code for the scene primitives is stored in a file called model.fx. This model.fx file has the code for both vertex shaders and fragment shaders.

GLSL shader code for the scene primitives is stored in two files model.vert and model.frag. The model.vert file has the code for vertex shader and the model.frag file has the code for the fragment shaders.

Shaders are implemented differently for OpenGL and DirectX. Hence there are different implementations of the Shader class for the different platforms. This is shown in Fig. 4.5.

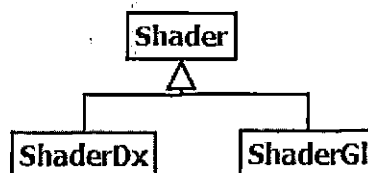


Fig. 4.5: The Shader Class

The Matrix class is used to set the world matrix and projection matrix. Matrices

are set differently in DirectX and OpenGL so there are different implementations of the Matrix class as shown in Fig. 4.6.

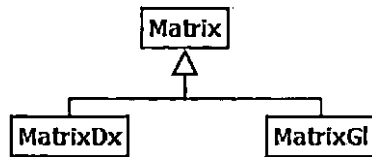


Fig. 4.6: The Matrix Class

The Texture class is used to load images for the scene primitives. It gets the x, y, normals nx, ny, nz and u,v co-ordinates for the image that are used to texture the scene primitive model. Textures are implemented using different derived classes for DirectX and OpenGL as shown in Fig. 4.7.

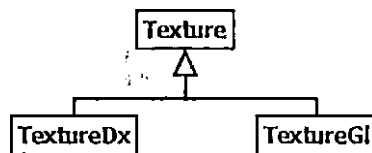


Fig. 4.7: The Texture Class

The Model class is used to load and render static 3D models.

Events on the scene primitives are handled in the same way as for the GUI primitives. To check if any scene primitive is clicked, a picking technique is used. This picking technique is explained in detail in chapter 5. When it is confirmed that a particular scene primitive is clicked, then the function passed to the SetScenePrimitiveHandler() is invoked.

The following C# code is an example of handling mouse click event on a scene primitive.

```
int scenePrimitiveHandle =
    Geng.CreateScenePrimitive(
```

```

        0, 0, 0, 0, -10, 0, -1, 1);
Geng.SetScenePrimitiveModel(
    scenePrimitiveHandle,
    "scene/tiger.geng_model");
Geng.SetScenePrimitiveHandler(
    scenePrimitiveHandle,
    HandleTigerClick);

public void HandleTigerClick(int handle)
{
    Geng.SetKeyHandler(null);
    Geng.UnloadGraphics();
    Program.Instance.ChangeState(null);
}

```

After creating an empty scene primitive and then telling the scene primitive to render a static 3D model, the `SetScenePrimitiveHandler()` function is called. This function associates the an integer handle for the scene primitive to a function. In this case the integer handle to the scene primitive is associated with the `HandleTigerClick` function. Hence when the user clicks on the scene primitive, the function `HandleTigerClick()` is called. The static 3D models are created in the 3D modeling applications Maya or Blender. These models have to be converted to the Geng specific format using the Geng exporters.

The following Fig. 4.8 shows how a scene primitive is rendered on the application window.

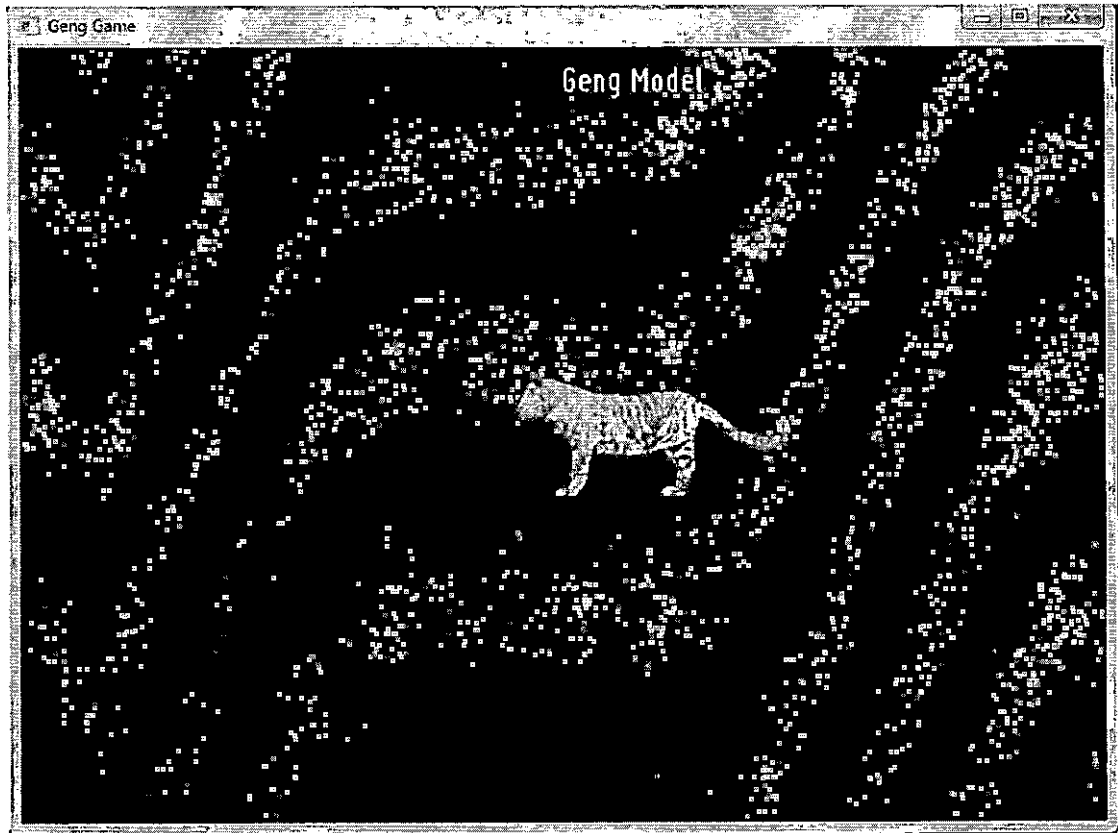


Fig. 4.8: Rendering Scene Primitives

## 5. PICKING

### 5.1. *Object Picking*

Object picking is a technique that determines if the user has selected a particular 3D object (or a 3D primitive) with the mouse. Picking is often a necessity in 3D games and applications where the user interacts with the 3D world using the mouse. An object picking technique is used in the Geng library to check if a Geng primitive such as a GUI primitive or a scene primitive is selected. The primitives in Geng are rendered in a particular order. First the scene primitives are rendered and then GUI primitives are rendered. Thus, when the user clicks on the application window, the following things happen.

- The Geng API functions first check if a GUI primitive is clicked. If a GUI primitive is clicked then the function corresponding to that GUI primitive is called.
- If a GUI primitive is not clicked, then the Geng API functions check if a scene primitive is clicked. If a scene primitive is clicked, then the function corresponding to that scene primitive is called.

Object picking is coded in a platform independent class called the Picker class. This class is shown in the Fig. 5.1.

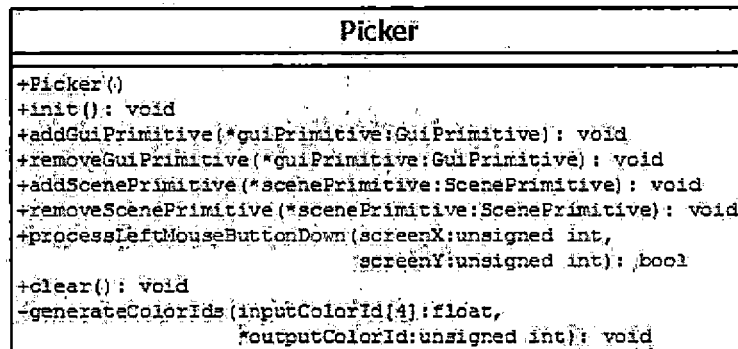


Fig. 5.1: The Picker Class

The Picker class has the following functions.

- void init()

This function initializes the Picker class. It creates a projection matrix, sets shaders, creates a vertex buffer and an index buffer for use in generating color IDs.

- void addGuiPrimitive(GuiPrimitive \* guiPrimitive)

This function generates an output color ID from a randomly selected input color ID. It makes sure that a unique input color ID is generated for each GUI primitive. Then it adds the GUI primitive to the list of pickable objects.

- void removeGuiPrimitive(GuiPrimitive \* guiPrimitive)

This function removes the GUI primitive from the list of pickable objects.

- void addScenePrimitive(ScenePrimitive \* scenePrimitive)

This function generates an output color ID from a randomly selected input color ID. It makes sure that a unique input color ID is generated for each scene primitive. Then it adds the scene primitive to the list of pickable objects.

- void removeScenePrimitive(ScenePrimitive \* scenePrimitive)

This function removes the scene primitive from the list of pickable objects.

- `bool processLeftMouseButtonDown(unsigned int screenX, unsigned int screenY)`  
This function checks if the event is consumed by objects click handler. Then it returns true otherwise it returns false.
- `void clear()`  
This function clears the list of pickable objects.
- `void generateColorIds(float inputColorId[4], unsigned int * outputColorId);`  
This is a private function of the Picker class. It is called by `addGuiPrimitive()` and `addScenePrimitive()` functions of the Picker class. It returns the output color ID for the given input color ID.

Whenever a Geng primitive is created, such as a GUI primitive, or a scene primitive, it is added to the map for the pickable objects.

## 5.2 Picking in Geng

Each pickable object in Geng is mapped to an input color ID and an output color ID. These are unique color IDs, which are generated when a primitive is created. The input color ID is randomly selected and the output color ID is generated using `generateColorIds()` function from the Picker class. The need for two color IDs come from the fact that shaders are used to render the primitives in Geng. For a given input color ID, shaders can render different colors. In order to get the output color ID, a picker shader renders a small triangle in the middle of the screen with the input color ID and then a pixel is picked from that triangle and its color is determined and returned as an output color ID. So every time a pickable object is created by the user application, it is added to the map of pickable objects and then it is mapped to an input color ID and output color ID. This is shown in the Tab. 5.1.



Object Name	float inputColorID [4]	unsigned int outputColorID
Object 1	Color01	Color05
Object 2	Color04	Color11

Tab. 5.1: Pickable Object Map

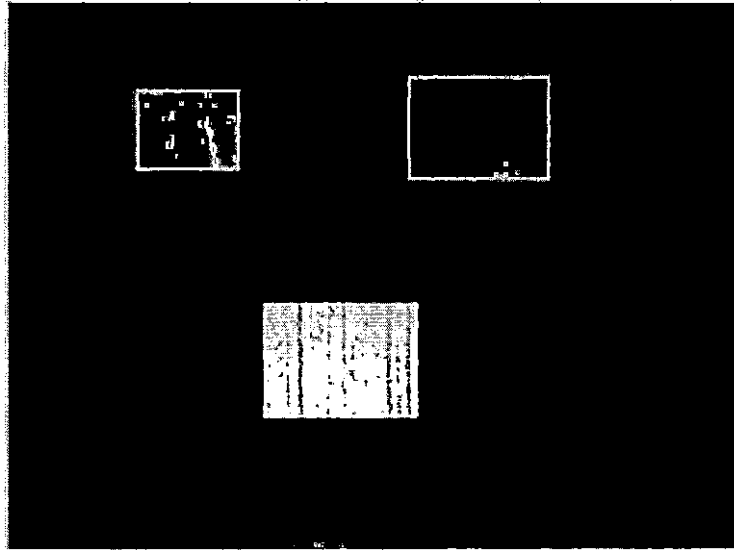
Assets are rendered in Geng using shaders. There are different types of shaders for different platforms. There are two types of rendering in Geng. These are listed as follows.

- Normal Rendering is the default type of rendering. Every asset is rendered with its original color and texture settings.
- Pick Rendering is triggered by a mouse click. In this type of rendering, only the pickable objects are rendered with the input color IDs from the pickable object map; non-pickable objects are rendered with the color pure black.

Whenever a mouse click happens on the application window, the following events happen:

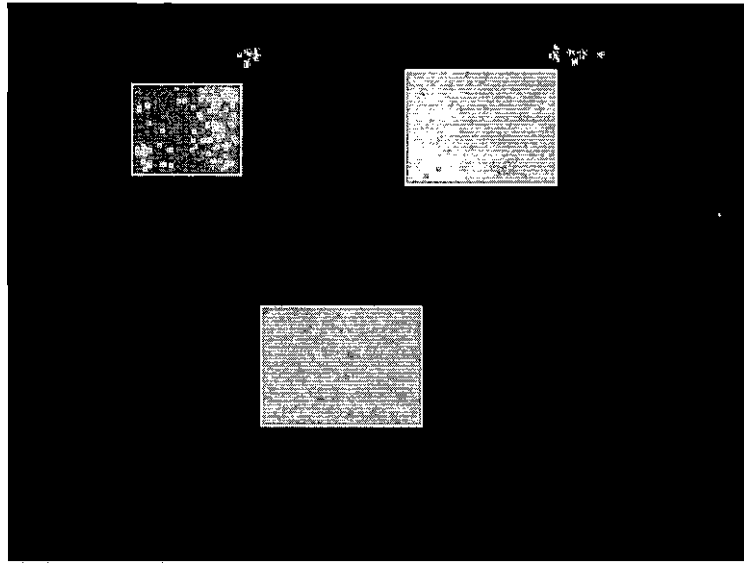
- The pick rendering is used for the Geng primitives.
- As a result of pick rendering, all the pickable objects are rendered with their input color IDs.
- Then, the color at the point on the application window, where the mouse click happened is obtained.
- Once the output color ID is acquired, the map of pickable objects is searched for that output color ID value.
- If an object corresponds to that particular output color ID, then the click event for that object is called.

Consider the Fig. 5.2. It shows how three GUI primitives are rendered with normal rendering. Each of these GUI primitives is pickable and hence has an input color ID and corresponding output color ID.



*Fig. 5.2: Normal Rendering*

Whenever the user clicks on the application window, pick render mode is triggered and each of the three GUI primitives are rendered with their input color IDs as shown in the Fig. 5.3. After the back video frame buffer is filled from a pick rendering operation, the system does not swap the video buffers, so the pick rendering has no visual effect except to possibly slow the frame rate by a small amount.



*Fig. 5.3: Pick Rendering*

Once it is determined which object is clicked, all the objects are rendered with normal rendering.

## 6. ART ASSET PROCESSING

### 6.1 *Shaders for Asset Rendering*

A shader is an executable program that runs on the graphics hardware. Shaders allow developers to write custom algorithms that can operate on the data that compose their virtual scenes. Shaders can be used to create just about any effect you can think of, which gives developers a high level of freedom and flexibility regardless of the graphics API being used. There are three types of shaders. They are listed as follows.

1. Vertex shader is the code executed on each vertex of the geometry passed to the graphics hardware. Input to the vertex shaders come from application itself. Vertex shaders perform calculations that are needed to be performed per vertex.
2. Geometry shaders sit between vertex shaders and pixel shaders. Once data is processed for vertex shaders, it is then passed to geometry shaders. These shaders are optional. The Geng library does not use geometry shaders.
3. Pixel shaders are also known as fragment shaders. These operate on each pixel of the geometry. The input to pixel shaders come from vertex shaders or geometry shaders.

Each of these shaders operates on various types of information. Combined together, they create one shader program. During rendering of a scene, only one shader can be active at a time. There are three shaders used in Geng.

1. Gui shader, to render the GUI for Geng.

2. Model shader, to render static 3D models.
3. Pick shader, to render models and Gui for picking.

#### Shader Languages:

In Geng, there are two shader languages used. These languages are listed as follows.

1. High Level Shader Language (HLSL), which is High Level shading language created by Microsoft for their DirectX graphics API.
2. OpenGL Shader Language (GLSL), which is OpenGL shading language for the OpenGL graphics API.

In the Geng project, `working_dir` is the default place where the Geng library functions look up the files. Every opened file is opened relative to this `working_dir`. The shader code is written in different files and should be present in the `working_dir`. This code is dynamically loaded from `working_dir` depending on the Geng build used.

For High Level Shader Language (HLSL) there are `.fx` files created for each shader. This file contains the code for both vertex and pixel shaders. The `.fx` files are called effect files. Effect Files are useful to separate art from the core graphics engine. Effect files have three parts: variable declarations, techniques and passes, and functions. Functions are shader code written in High Level Shader Language (HLSL). Techniques and passes define the rendering techniques. These rendering techniques can be chosen based on the hardware specifications.

For OpenGL Shader Language (GLSL), there are two shader files; one for each shader. By convention, for vertex shader there is a `.vert` file and for fragment shader there is a `.frag` file.

The code to load the shaders is called in the `init()` function of the Graphics class. According to the Geng build, corresponding code from the `GraphicsDx` and `GraphicsGL` classes are called.

## 6.2 Geng Asset Conversions

Assets are the building blocks of any game or simulation. Assets can be images or static models that need to be rendered in the game. These assets should be converted into the Geng specific file format. Having the Geng specific file formats for the assets, makes it easier to render them in any platform. Also, it moves the code complexity to the exporters, keeping the Geng library implementation simpler. In the Geng project, `working_dir` is the folder that contains the files needed in the working directory when the program runs. Every opened file is opened relative to the working directory. The working directory has various sub folders for assets such as fonts, GUI primitives, and scene primitives etc. All the converted assets should be placed in the appropriate folders in the working directory. Geng has two types of tools that convert assets to the Geng compatible file format: an image converter and two static model exporters.

The image converter is a C++ program that runs on the command line. It converts images in various formats to the Geng specific image file format. Fig. 6.1 shows the `geng_image` file format.

geng_image	
<i>&lt;Height&gt;</i>	Image height
<i>&lt;Width&gt;</i>	Image width
<i>&lt;Padded width&gt;</i>	Padded height
<i>&lt;Padded Height&gt;</i>	Padded width
<i>&lt;Pixel data&gt;</i>	Pixel data in RGBA format

Fig. 6.1: Geng\_image File Format

In the Fig. 6.1, image height and width are the actual height and width for the given image. Padded height and width are the extra width and height added to the actual width and height to make them into nearest powers of 2. For example, consider an image file with actual width equal to 28 and actual height equal to 14; the padded

width would be calculated as 32 and padded height as 16.

The static model exporters are plug-ins written in python that export static models from 3D modeling applications such as Maya and Blender to the Geng specific file format. Fig. 6.2 shows the Geng specific static-model file format.

<i>geng_model</i>	
<i>&lt;No of vertices&gt;</i>	No of vertices for the model
<i>&lt;Vertex data&gt;</i>	Vertex data for the model
<i>&lt;No of indices&gt;</i>	No of indices for the model
<i>&lt;Index data&gt;</i>	Index data for the model
<i>&lt;No of sub-meshes&gt;</i>	No of sub-meshes for the model
<i>For each sub-mesh.</i>	
<i>&lt;Texture file name&gt;</i>	Texture file for each sub-mesh
<i>&lt;Vertex start&gt;</i>	Start vertex for each sub-mesh
<i>&lt;Vertex count&gt;</i>	No of vertices for each sub-mesh
<i>&lt;Face start&gt;</i>	Start face for each sub-mesh
<i>&lt;Face count&gt;</i>	No of faces for each sub-mesh

Fig. 6.2: Geng\_model File Format

A model is created using a mesh. A mesh is a shape made up of polygons connected together. A mesh can be made up of several connected sub-meshes, where each sub-mesh is made up of several polygons connected together. In the Fig. 6.2, number of vertices is the total number of vertices for the complete mesh. Vertex data is data for each vertex such as x, y, z co-ordinates, normal vector, color. The number of indices is the total number of indices to render the complete mesh. Index data is the integer data that specifies the order in which the vertices should be rendered. Now the whole

mesh for the model can be a combination of several sub-meshes. The number of sub-meshes gives the total number of sub-meshes that were combined to form a mesh for the model. The information about each sub-mesh is stored in order. The texture file name is the name of the image file that textures a particular sub-mesh. Vertex start is the starting vertex for that particular sub-mesh. Vertex count is the number of vertices for that particular sub-mesh. Face start is the starting polygon number for that particular sub-mesh. Face count is the number of polygons in that particular sub-mesh. Design standards for the static models:

The following are the design standard should be followed while creating static 3D models in any of the 3D modeling applications.

- The static model should be in the form of one mesh. There can be several sub-meshes but all should be joined together to form a single mesh.
- The main mesh for the model should have at least one uv mapped texture.

### 6.3 Image Converter

Image converter, also called asset conversion tool, for the Geng library is written in C++. It converts images from bitmap, targa, JPEG formats to the Geng specific format. These images can be used to render GUI images. This converter is supposed to run on the windows operating system. After conversion, the loading and rendering of assets is platform independent. The asset conversion tool is a command line utility. This utility can be used as follows.

To convert an image the following command should be used:

```
assets.exe image <image filename>
```

This converts the image filename into a Geng compatible image file with an extension geng\_image.



## 6.4 Exporter for Maya

Maya is a 3D modeling application being developed by Autodesk Inc [7]. Maya is used to generate 3D assets for game development. It has comprehensive tools for modeling, animation, visual effects, and rendering solution [7]. The Geng Maya exporter converts the static 3D model created in Maya to the Geng specific file format.

Maya Software Development Kit (SDK) functions can be used to create custom shaders and nodes in maya. The SDK includes a C++ API that provides functionality for querying and changing the Maya model [7]. In addition, the SDK contains Python bindings to the Maya API [7]. These bindings allow to call the Maya API from Python. This is called as Maya Python API [7].

Using any of these APIs, you can add new elements to Maya such as: shapes, shaders, commands etc. The exporter for Geng is written in Python using Maya Python API. At the lowest level, Maya stores all the graphical information in Dependency Graph (DG). Information in the DG is stored in objects called nodes. Nodes have properties called attributes that store the configurable characteristics of each node.

The Maya exporter uses the following classes the Maya Python API.

The `MFnMesh` class is a mesh class that provides access to polygonal meshes. This is one of the main classes of Maya. An example of this class is shown in the following sample code.

```
mesh.getVertices( verts, vertexList )
```

This function gets the list of vertices for all the polygons in the mesh.

The `MItSelectionList` is a class used to iterate over the items in the selection list (`MSelection`). A selection criteria can be specified so that only those items of interest on a selection list can be obtained. If a criteria is specified then the children of DAG will be searched if the selection item does not match the selection criteria. The following code shows an example of the class `MItSelectionList`.

```

selectionList = OpenMaya.MSelectionList()
OpenMaya.MGlobal.getActiveSelectionList( selectionList )
itList = OpenMaya.MItSelectionList( selectionList, OpenMaya.MFn.kTransform )
while not itList.isDone():
    <code>
    -----
itList.next()

```

In the code above, an iterate itList is created for the filter equal to OpenMaya.MFn.kTransform.

The MFnDagNode class is the DAG node Function Set. It provides methods for attaching Function Sets to DAG nodes, querying, and adding children to DAG nodes. The following code shows an example of the MFnDagNode class.

```

fnDagNode = OpenMaya.MFnDagNode( path )
numChildren = fnDagNode.childCount()

```

In the example above, the MFnDagNode class is used to get the child count for the DAG node.

The MItDependencyNodes class is a dependency Node iterator. It is used to traverse all the nodes in Dependency Graph. A selection criteria can be provided to select a particular node. The following code shows an example of the MItDependencyNodes class.

```

itDN = OpenMaya.MItDependencyNodes( OpenMaya.MFn.kDependencyNode )
while not itDN.isDone():
    <code>
    -----
itDN.next()

```

In the example above dependency node iterator is used with filter as OpenMaya.MFn.kDependencyNode

To run the Geng Python script in Maya, the following steps should be followed.

- Create static 3D model.
- Open Script Editor.
- In the Python tab write the following commands:

```
import GengExport
GengExport.UI()
```

Where the GengExport is the Python exporter script name and UI() is the entry level function name in Python exporter script.

The following is an example to show how to create a simple static 3D model in Maya.

Fig. 6.3 show the Maya Application window.

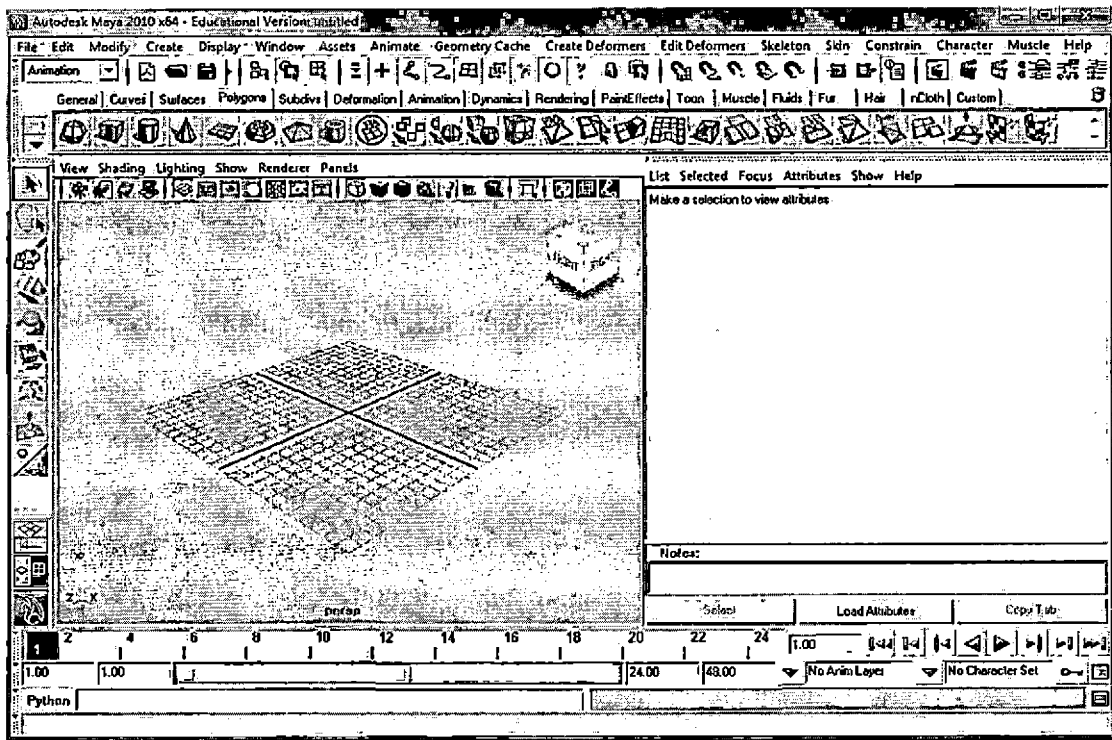


Fig. 6.3: Maya Application

Using the range of primitives in Maya, such as polygon sphere and polygon cube,

in combination with all the transformation tools, create a mesh model. Fig. 6.4 shows a screenshot of mesh created using polygon torus primitives.

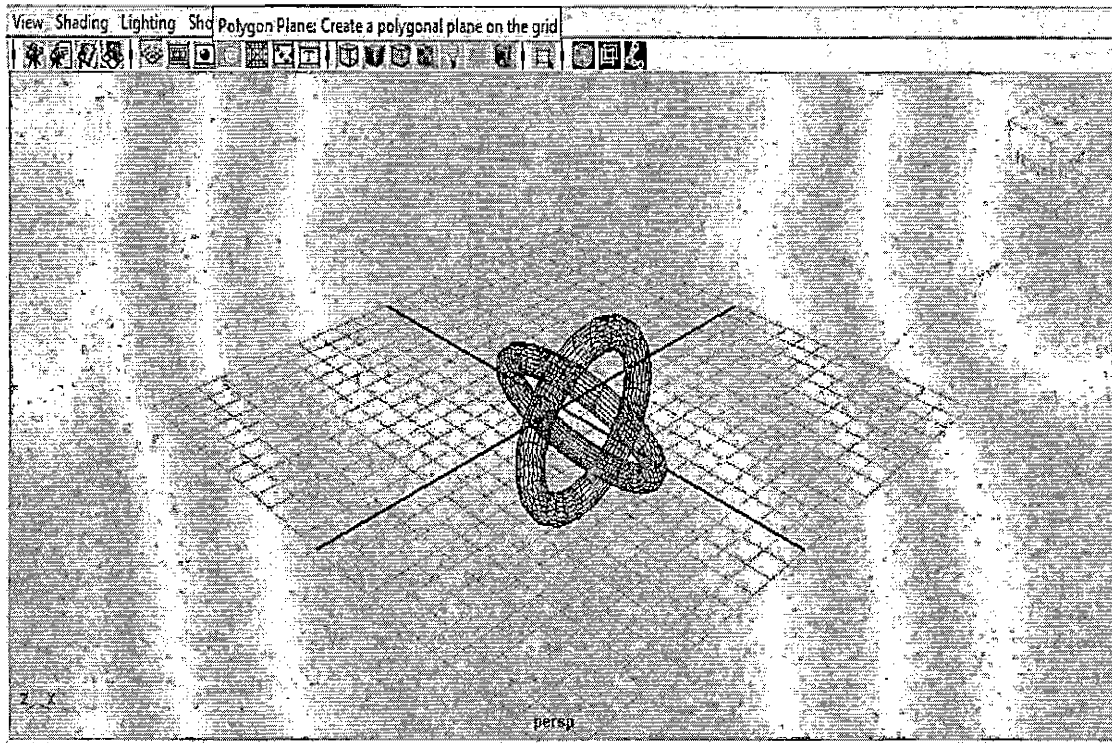


Fig. 6.4: Maya Application Creating a Mesh

After creating the mesh, it should be textured. To texture a mesh in Maya, assign a material to the mesh. After assigning a material, in the materials menu, from common materials attribute, select color. A menu to select texture will be displayed as shown in Fig. 6.5. Select File option and provide the texture file name.

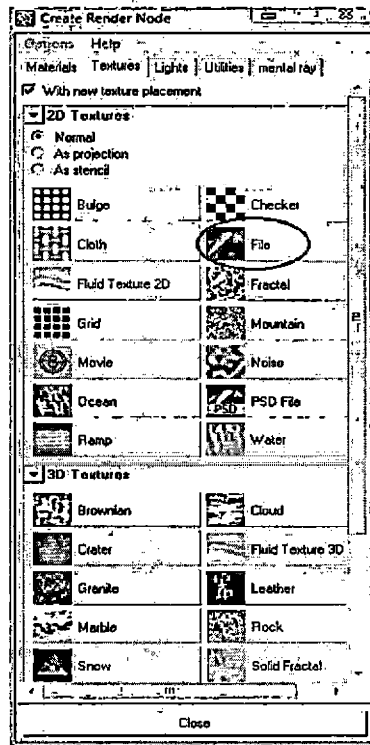


Fig. 6.5: Select Texture File

After assigning the texture file name, select the textured button from the menu which will texture the whole mesh with selected texture as shown in Fig. 6.6

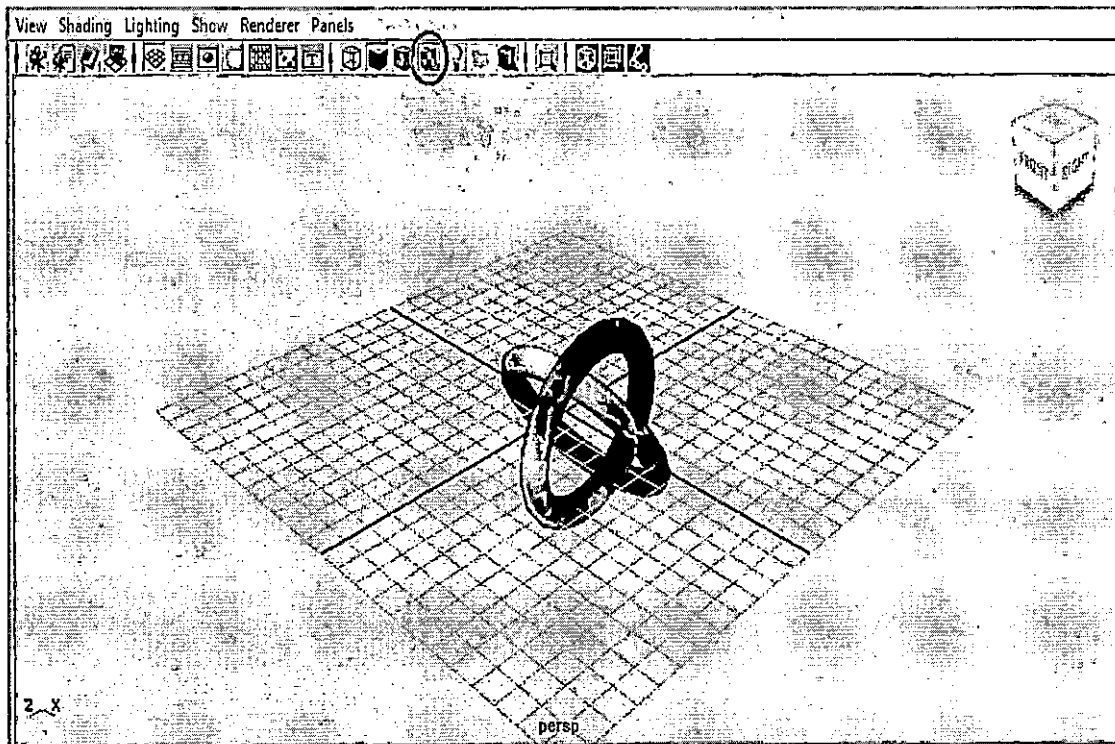


Fig. 6.6: Maya Application Texturing a Mesh

To export this model, click on the script editor button on the bottom right corner. The script editor will appear. In the Python tab write the following commands:

```
import GengExport
GengExport.UI()
```

A file Save As box will appear with default file extension as geng\_model. Type the name for the model and the model will be converted into geng\_model format along with the texture file in geng\_image format.

### 6.5 Exporter for Blender

Blender [1] is a 3D modeling application. It is an open source available for all major operating systems [1]. Blender can be used to create models, shaders, animation and

interactive 3D [1]. Exporter for Blender is written in Python. The Blender Python API is used.

Blender uses an object oriented architecture [1]. Blender objects and their attributes are same for both, Blender Python API and the user interface (the GUI). This makes it easy to understand the Blender objects and attributes. Each Blender graphic element is composed of two parts: an Object and Object Data [1]. The Object holds basic information of an object like its size, position etc. This is common information for all the objects. The Object Data holds information specific to that particular type of element and which is not common for all the objects. Each Object has a link to its associated Object Data. A single Objects Data may be shared by many objects [1]. All the Blender objects have a unique name. The Blender Python API provides the functions can create user interface elements such as menus and panels [1]. The Blender Python API also provide functions that can edit data such as meshes and scenes. Blender Python API has top module and sub-modules that have functions that can be used to manipulate Blender data [1].

The Geng Blender exporter uses the following modules from the Blender Python API

The Blender.Draw sub-module provides access to a windowing interface in Blender [1]. This include many kinds of buttons such as push, toggle, menu, number, string, slider, scrollbar [1]. It also supports string drawing. The following is the example code to illustrate how to use this module.

```
pupMenuResult = Blender.Draw.PupMenu("Selected object is not a mesh.%t|OK")
```

The PupMenu method from Blender.Draw sub-module is used to create a message box.

The Blender.sys sub-module provides a set of functions called as helper functions [1]. These functions are mostly related to the files operations. The following is the example that shows how to use this sub-module.

```
fname = Blender.sys.makename(ext = ".geng_model")
```

The `makename` function from the `Blender.sys` sub-module is used to force “geng.model” to be the extension to exported files.

The `Blender.Window` module provides access to Window functions in Blender. The following is the example that shows how to use this module.

```
Blender.Window.FileSelector(self.on_file_select, "Export Geng", fname)
```

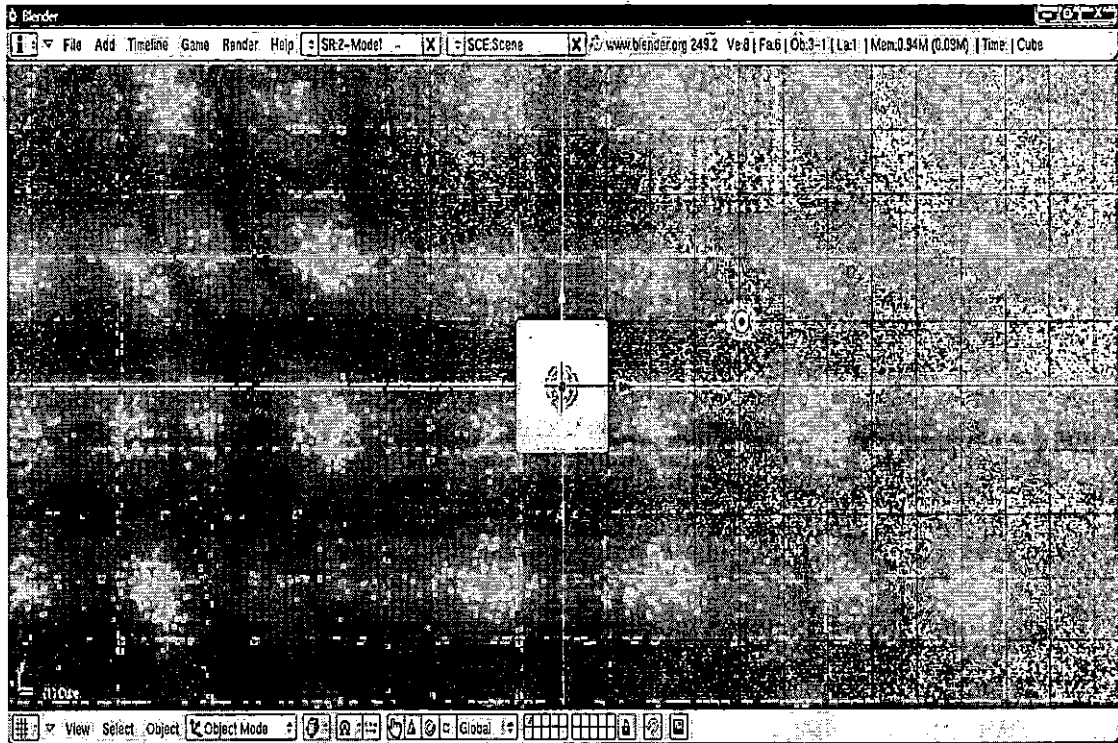
`FileSelector` is a function from the `Blender.Window` sub-module. It is used to open the file selector window in Blender. After the user selects a filename, it is passed as a parameter to the function callback given to `FileSelector()` [1].

The `Blender.Mesh` module provides access to Mesh Data objects in Blender.

The following procedure is an example to show how to create a simple static 3D model in Blender. This procedure creates a textured cube.

- The Blender application provides a lot of mesh primitives that can be used to create various shapes and objects. To create a cube, add a primitive cube mesh to the Bender application. This is shown in Fig. 6.7.





*Fig. 6.7: Blender Application*

- After creating the mesh, it should be textured. To texture a mesh in Blender, perform the following steps.
  - Change the mode from object mode to edit mode.
  - Then, from the mesh menu, select option UV Unwrap as shown in the Fig. 6.8.

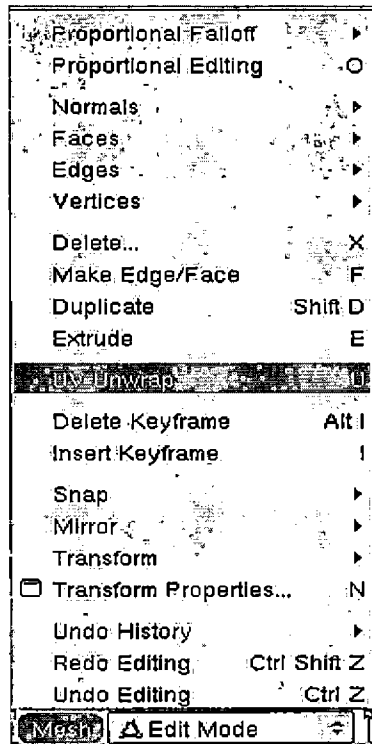


Fig. 6.8: Mesh Menu

- In the split view, select UV Image editor view. This is shown in Fig. 6.9. This mode will give an access to the image menu. Load the texture image to texture the mesh as shown in Fig. 6.10.

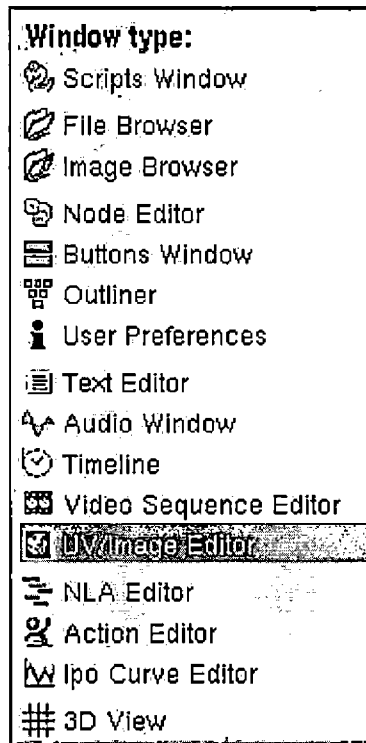


Fig. 6.9: UV Image Editor

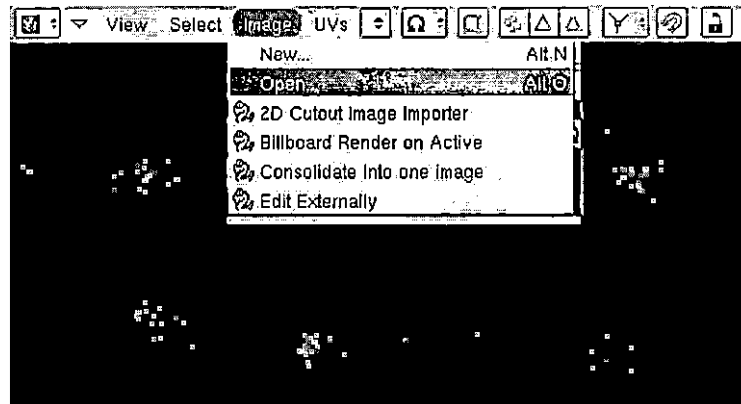


Fig. 6.10: Load Texture

- Once the texture is selected, to apply it to the whole mesh, Set the Draw type to Textured and change the mode to object mode as shown in Fig. 6.11.

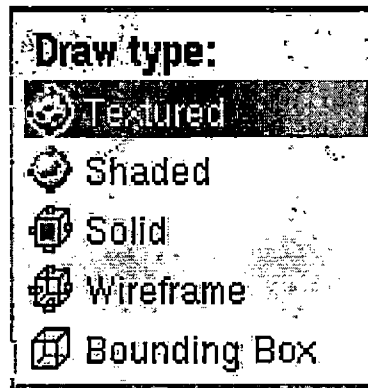


Fig. 6.11: Draw Type Textured

- After the model is created and ready to be exported to Geng, Select the menu option File and Export and export it as geng\_model as shown in Fig. 6.12.

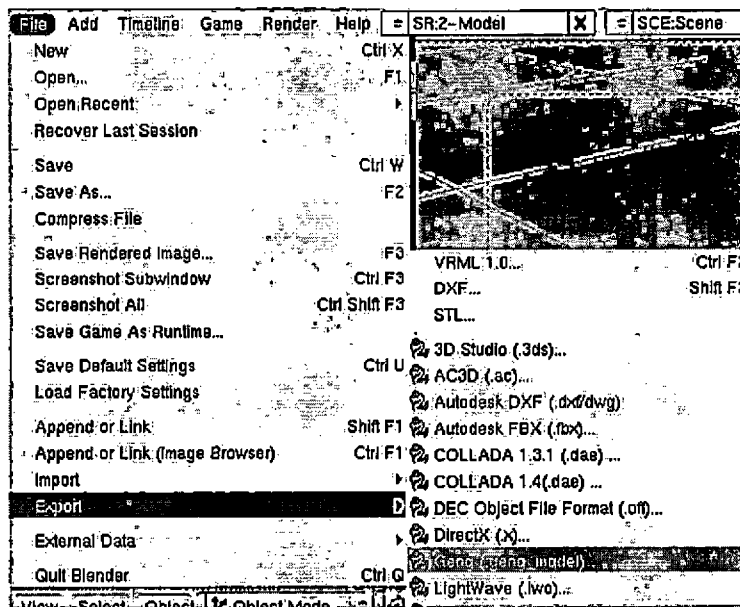


Fig. 6.12: Export

## 7. GENG LIBRARY USAGE

### 7.1 Geng Initializations

The Geng library can be used to create a 3D game or simulation. The user application can be written in languages such as C++, C#, Java, Python or any other language that integrates with libraries that have C linkage. Geng Game is the example user application written in C#. The Geng API provides a set of functions that need to be imported by the user application. These functions in turn call OpenGL or DirectX depending on the build of the Geng library.

The syntax for importing the functions provided by the Geng API in C# is shown below:

```
[DllImport (dllFilename)]  
public static extern <return type> funcName(parameter list);
```

The dllFilename is the Geng build file name for the platform being used. For example, for Windows Directx build, the Geng build filename is "geng\_win\_dx.dll". The extern modifier is used to declare that the method is implemented externally. The Geng API is written in C. To import a function from the C API into the C# code, it is declared as static.

To handle events on the GUI primitives, callback functions are used. For this, a delegate needs to be defined. Below is the example code that creates a delegate called GuiPrimitiveHandler for GuiPrimitives.

```
public delegate void GuiPrimitiveHandler(int guiPrimitive);
```

The Geng API provides a set of functionality to create an application window, unload graphics etc. These functions are called only once during a game life-cycle.

The sample code below shows how to create an application window using the Geng API function.

```
Geng.InitWindowed(800, 600, "Gaming Application");
```

The API function `InitWindowed()` creates the application window of 800 pixels width and 600 pixels height and with the title "Gaming Application".

The Geng API also provides a very important function called `Tick()`. This function processes user input events and generates video frames. This function can be used by the user application in a loop and should keep it running until the user exits the application.

## 7.2 Geng Graphics

GUI primitives can be used to render different effects such as text, images or both. Following code sample shows how to use a GUI primitive to display text.

```
int titleHandle = Geng.CreateGuiPrimitive
    (Geng.ScreenPositionUpperCenter,
    0, -40, Geng.True);
Geng.SetGuiPrimitiveLabel(
    titleHandle, "fonts/miso_32",
    0, "Welcome", 0, 0);
```

`CreateGuiPrimitive()` function creates an empty GUI primitive with the screen position and x and y offsets and visible equal to true. It returns handle for that particular GUI primitive. This handle is passed to the `SetGuiPrimitiveLabel()` function along with the font and string that needs to be rendered. The `SetGuiPrimitiveLabel()` function tells the GUI primitive to render string. The above code renders the string "Welcome" for the given font `miso_32`.

The following code sample shows how to use GUI primitive to render image.

```
int woodHandle = Geng.CreateGuiPrimitive(Geng.ScreenPositionDeadCenter,  
    0, 0, Geng.True);  
Geng.SetGuiPrimitiveImage(woodHandle, "gui/wood_64_64.geng_image");
```

CreateGuiPrimitive() function creates an empty GUI primitive with the screen position, x and y offsets and visible equal to true. It returns handle for that particular GUI primitive. This handle is passed to the SetGuiPrimitiveImage() function along with the image that needs to be rendered. The SetGuiPrimitiveImage() function tells the GUI primitive to render an image. In this case the image wood.64\_64.geng\_image would be set for the GUI primitive. The image can have events such as click that can be handled using the Geng API function. This is explained in the next section.

Scene primitives can be created to render 3D static models. The code to render the scene primitives is as follows.

```
int scenePrimitiveHandle = Geng.CreateScenePrimitive(  
    0, 0, 0, 0, -10, 0, -1, 1);  
Geng.SetScenePrimitiveModel(scenePrimitiveHandle, "scene/tiger.geng_model");
```

CreateScenePrimitive() creates an empty scene primitive with the x, y, z positions and visible equal to true. It returns the handle to that particular scene primitive. This handle is then passed to the SetScenePrimitive() function along with the model that needs to be rendered. The SetScenePrimitive() function tells the scene primitive to render a static 3D model.

### 7.3 *Event Handling*

The Geng API functions can be used to handle input events. Event handling is done using callback functions. A callback function is a function that is called through a function pointer. In Geng, function pointers are used to call the respective event

handler function for the GUI and scene primitives. From a C# program, a delegate must be declared and used as a data type for the callback function. The following is an example of a delegate that can be used to set GUI primitive handlers.

```
public delegate void GuiPrimitiveHandler(int guiPrimitive);
```

This delegate takes one argument, which is the integer handle to the GUI primitive that was clicked. A function whose signature matches this delegate is passed as an argument to the Geng API function `SetGuiPrimitiveHandler()`. The following is the C# declaration of `SetGuiPrimitiveHandler()`.

```
[DllImport(dllFilename)]
public static extern void SetGuiPrimitiveHandler
    (int guiPrimitive, GuiPrimitiveHandler guiPrimitiveHandler);
```

When a GUI primitive is created, an event handler function can be created by the user application. The following code is an example of defining and registering a GUI primitive event handler function.

```
int woodHandle = Geng.CreateGuiPrimitive(Geng.ScreenPositionDeadCenter,
    0, 0, Geng.True);
Geng.SetGuiPrimitiveImage(woodHandle, "gui/wood_64_64.geng_image");
Geng.SetGuiPrimitiveHandler(woodHandle, HandleTitleClick);

public void HandleTitleClick(int handle)
{
    Geng.SetKeyHandler(null);
    Geng.UnloadGraphics();
    Program.Instance.ChangeState(null);
}
```

The `HandleTitleClick()` is passed as an argument to the `SetGuiPrimitiveHandler` function. Whenever, user clicks on this particular GUI primitive, `HandleTitleClick()`



function is called. The events on Geng Primitives and any other user input events are handled in the same fashion as shown above.

## 7.4 *Geng Physics Library*

The Bullet Physics Library is a library of functions that can do 3D collision detection and rigid body dynamics for games and animation. The Bullet Physics Library has a collection of components for collision detection, rigid body and soft body dynamics [2].

The Geng Physics library uses three main components from the bullet physics engine: BulletCollision, BulletDynamics and LinearMath. The Geng Physics library is a wrapper around the Bullet Physics Library. It uses the functions of the Bullet Physics Library to create custom bodies and objects that can be used by the Geng Game application. The Geng Physics library provides an API. The functions of this API can be imported by the user application.

```
PS_API void PhysInit();
```

This function initializes the physics system, creates the ground and rigid bodies.

```
PS_API void PhysShutdown();
```

This function deletes the physics system created by PhysInit().

```
PS_API void PhysTick(float dt);
```

This function generates the next step for the physics simulation. This function should be called in the loop by the user application.

```
PS_API int PhysCreateBoxShape(  
    float xHalfExtent,  
    float yHalfExtent,  
    float zHalfExtent);
```

This function creates a bounded body in shape of a box with the given x, y and z values.

```

PS_API int PhysCreateStaticBox(
    int boxShapeHandle,
    float axisX, float axisY, float axisZ,
    float angle, float x, float y, float z);

```

This function creates a static rigid body for the given box shaped handle. This static rigid body is then added to the dynamics world. This static box is then transformed to the position given by the rotation axis x, y, z and angle of rotation and positions x, y and z. Static rigid bodies have infinite mass. They cannot move once placed in a particular position. However they can participate in collision.

```

PS_API void PhysDestroyStaticBox(int staticBoxHandle);

```

This function removes the given static rigid body from the dynamics world and deletes the static rigid body object.

```

PS_API int PhysCreateDynamicBox(
    int boxShapeHandle,
    float axisX, float axisY, float axisZ,
    float angle, float x,
    float y, float z,
    float mass,
    TransformUpdateHandler transformUpdateHandler);

```

This function creates dynamic rigid bodies for the given boxShapeHandle. Dynamic rigid bodies have mass and for every simulation frame when the dynamic rigid body moves, its world transform is updated. These dynamic rigid bodies are then added to the dynamics world.

```

PS_API void PhysDestroyDynamicBox(int dynamicBoxHandle);

```

This function removes the given dynamic rigid body from the dynamics world and deletes the dynamic rigid body object. These functions provided by the Geng Physics

API can be used by the user application Geng Game to create various visual effects such as flying boxes and collisions, etc.

The following Fig. 7.1 shows the Entity class hierarchy in the Geng Game application. These classes make use of the functions provided by the Geng Physics API.

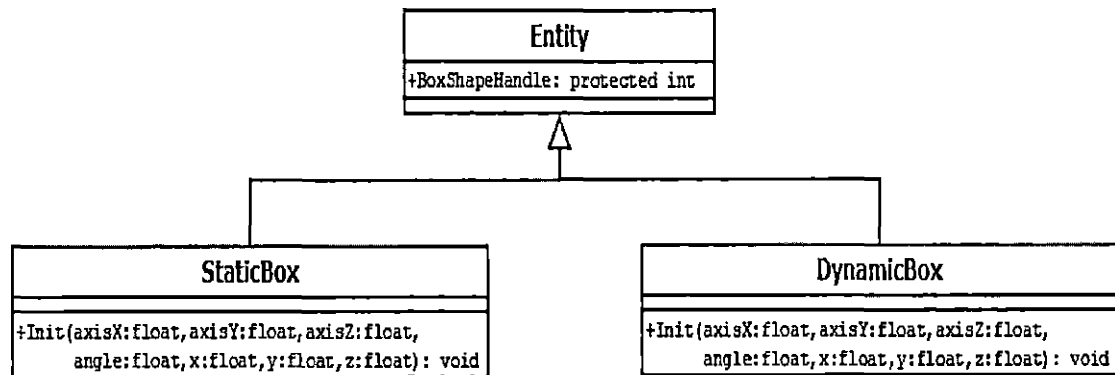


Fig. 7.1: Entity Class Hierarchy

The Entity class uses the `PhysCreateBoxShape()` function from the Geng Physics API and updates the `BoxShapeHandle`. The `StaticBox` class inherits the Entity class and uses the Geng Physics API function `PhysCreateStaticBox()`, passing a `BoxShapeHandle` to create a bounding volume for the Geng Primitive. The `DynamicBox` class inherits the Entity class and uses the Geng Physics API function `PhysCreateDynamicBox()`, passing a `BoxShapeHandle` to create a bounding volume for the Geng Primitive. The `StaticBox` and `DynamicBox` classes can be used to create bounding volumes for the Geng Primitives as shown in the code below:

```

List<StaticBox> staticBoxes = new List<StaticBox>();
for (int i = 0; i < 16; ++i)
{
    float x = -100 + (float)random.NextDouble() * 200;
    float y = 10 + (float)random.NextDouble() * 200;

```

```

float z = -100 + (float)random.NextDouble() * 200;
StaticBox box = new StaticBox(0, 1, 0, 0, x, y, z);
box.PickHandler = steeringPlayerController.PickHandler;
staticBoxes.Add(box);
}

```

The above code shows how to create a list of static boxes at random x, y, z positions.

```

StaticBox box = new StaticBox(0, 1, 0, 0, x, y, z);

```

When a StaticBox is created, the Init() function from StaticBox class is called. The code for the Init() function is as follows.

```

private void Init(float axisX, float axisY, float axisZ,
float angle, float x, float y, float z)
{
    this.x = x;
    this.y = y;
    this.z = z;
    scenePrimitiveHandle = Core.CreateScenePrimitive(
axisX, axisY, axisZ,
angle, x, y, z, Core.True);
    Core.SetScenePrimitiveModel(
        scenePrimitiveHandle, "scene/model.geng_model");
    physicsHandle = Physics.PhysCreateStaticBox(
        BoxShapeHandle, axisX, axisY,
axisZ, angle, x, y, z);
}

```

The code in Init() creates a bounding volume for the scene primitive.

## 8. CONCLUSION AND FUTURE DIRECTION

### 8.1 *Conclusion*

Geng is a platform independent library, which can be used to create graphical effects in games and scientific simulations. The Geng library provides an API that can be used by applications. The user of this API does not need to think about the underlying platform or write complex code to create graphical effects such as rendering images, text, or 3D models. The visual effects for the user application are the same irrespective of the underlying platform. The Geng library provides the user with two plug-ins for the 3D modeling applications Maya and Blender. This makes it easy for the user to import 3D models. Thus, game asset rendering is simplified because of these plug-ins. During the development of the Geng library, a lot of attention was paid to developing and adhering to organizational principles. This helped in increasing the maintainability of the code and should make future development a lot easier. The Geng library has no external dependencies on third party applications for handling user events or creating application windows. This makes the Geng library flexible. Currently, the Geng library provides support for the Bullet Physics Library through the Geng Physics API. The Bullet Physics Library can be used to do rigid and soft body dynamics and collision detection. These are very important features for any interactive game.

This iteration of Geng has the following advantages.

- The restructuring makes the Geng library maintainable and extensible.

- Removing additional dependencies on the third party applications make the Geng library more flexible.
- The exporters give the user application, a flexibility to create assets.

This iteration of Geng has the following disadvantages.

- In this iteration, more emphasis was on re-factoring and re-structuring. As a result all the features from the previous iterations are not supported by this iteration. The Geng library does not support animated models, which were supported by the previous iteration.
- As a result of removing the dependency on the Simple Direct Media Layer (SDL) library, the Geng library has to maintain more code.

## 8.2 Future Direction

In spite of all the functionality that the Geng library provides, it needs some more work to make it robust. The Geng library needs the following functionalities:

- Exporter for animated models.

Currently Geng provides an exporter for static 3D models. There are two exporters written in Python for Maya and Blender. However, Maya and Blender can also be used to create animations. Animation is a very important part of any 3D game application. Geng needs exporters that would convert animations created by Maya and Blender into the Geng compatible file formats. This also requires a change in the Geng library functionality, and new shaders will have to be coded to render the animated 3D models.

- Exporter for the particle systems.

Particle systems are used for modeling fuzzy objects such as water, cloud, fire, smoke, etc. These objects do not have well defined shapes and boundaries. Such

effects are desirable in gaming applications. The 3D modeling applications Maya and Blender can be used to create these effects. Geng needs exporters to convert particle systems created by Maya and Blender to the Geng specific format. In addition to these exporters, Geng needs changes to the Geng library to add the functionality to render the particle systems. Also new shaders should be created to render the particle systems.

- Shadows.

Shadows are an important part of a game. Shadows make any scene more realistic. Currently shadows are not a part of the Geng library. The scene primitives currently do not have any shadows. However in future, this feature needs to be added to the Geng library so that every scene primitive can render its shadow.

- Complete refactoring of the Geng library.

Geng coding standards are still evolving. Geng coding standards are merged with some of the coding standards provided by Google's style guide for C++. The Geng library needs a complete refactoring to adhere to these evolving coding conventions.

- Expand the example user code to a complete game.

Geng Game is an example user application written in C# to demonstrate the capabilities of the Geng library. However, this Geng Game application can be enhanced and expanded to create a complete game.

### 8.3 *Definitions, Acronyms, and Abbreviations*

- 3D: Abbreviation for three-dimensional computer graphics which are graphics that use a three-dimensional representation of geometric data.
- AI: Artificial Intelligence.

- GUI: Graphical User Interface.
- Geng: Game Engine being developed at CSUSB.
- HLSL: High Level Shader Language for DirectX.
- GLSL: OpenGL Shader Language.
- Shader: Set of software instructions which are used primarily to calculate rendering effects on graphics hardware with a high degree of flexibility.
- HUD: Stands for Heads up display.
- CG Shader: CG is shading languages created by Nvidia.
- UV Mapping: UV Mapping is a process texturing 3D models with 2D images.
- Mesh: Mesh is a shape made up of polygons connected together.
- Vertex Buffer: Vertex buffer holds information about a vertex such as x, y, z co-ordinates, normal vector, color etc.
- Index Buffer: Index buffers store index data. Indices are integer offsets into the vertex buffers.
- SDL: Simple Direct Media Layer. SDL is an open source library that provides interface to graphics, sound and input devices.
- Bullet Physics: Bullet Physics is an open source physics library that is used in game programming for the visual effects.
- FMOD: It is an audio library used in gaming applications to provide audio functionality.
- Blender: Blender is an open source 3D modeling application.
- Maya: Maya is 3D animation visual effects software developed by Autodesk.



- DAG: Directed Acyclic Graph.
- Python: Python is a powerful scripting language that is used in a wide variety of application domains.

## REFERENCES

- [1] Blender. <http://www.blender.org/>.
- [2] Bullet physics library. <http://bulletphysics.org/wordpress/>.
- [3] Calling dll functions. <http://msdn.microsoft.com/en-us/library/be80xase.aspx>.
- [4] Fmod. <http://www.fmod.org/>.
- [5] Geng website. <http://cse.csusb.edu/geng/>.
- [6] Google style guide. <http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>.
- [7] Maya. <http://usa.autodesk.com/maya/>.
- [8] Chris Ballinger. Mythic game project addition of artificial intelligence and quest system components. *CSUSB Masters Project*, June 2010.
- [9] Dave Shreiner et. al. *The OpenGL Programming Guide - The Redbook*. Addison-Wesley Publishing Company, 2006.
- [10] A. Sherrod. *Game Graphics Programming*. Course Technology PTR, 2008.
- [11] David Stover. An open source graphics engine for three-dimensional video games. *CSUSB Masters Project*, June 2010.