# Graphics Workflow for a Side-scrolling Game

*Collaboration with a colourist*

Omar El Sawi
National Centre for Computer Animation
Bournemouth University

## ABSTRACT

This project aims to explore and find a suitable way of creating environments for a proposed side-scrolling adventure game.

It aims to create a successful collaboration through the internet with a colourist, and dealing with both the technical and visual challenges of creating the data in a format suitable for display in a real-time application.

## 1 INTRODUCTION

Developing game graphics has both an aesthetic and technical side. In the early days of video-gaming, all roles were tackled by a lone programmer.

However, as greater complexity was required, a game team model emerged similar to motion pictures -where each game is the result of many specialists pooling their energies (Katz, Yates, 1996, p27).

This project is an attempt to explore the dynamics of collaboration; joining with Italian colourist Romina Denti, in order to take her work beyond still images, and into the context of a side-scrolling game.

During this collaboration several challenges were encountered of personal, visual and technical nature. By research of techniques employed through video-games, we attempt to find the right workflow for use in future game projects.

## 2 THE SIDE-SCROLLER

Side-scrolling games were particularly popular in the pre 32-bit era of video games, and had a quick learning curve, as the game mechanic restricted primary movement to left and right.

With the emergence of 3D games, and the extra dimension, side-scrolling games have become less popular. However, one might argue that the shift to 3D happened too fast and that there are still unexploited forms of expression where the two intermingle.

In our case, using a scroll as a form of representing our world, served our purpose of visual communication, and to confine the project within our means.

The concept of the scroll existed long before video games. In the Far East, the Japanese developed the storytelling potential of painting through the narrative scroll. And the Chinese made a great contribution to the history of painting through the Chinese landscape (Britannica, 2008).

The huge scroll was viewed one area at a time, and moving through it gave the sense of travelling through time and space.

*[IM1] This image shows the first quarter of the scroll painting, Along the River During Qingming Festival by Zhang Zeduan. In contrast to western art which depicts a specific situation, the Chinese scroll painting show a development in time. The scroll must be viewed not all at once, but section by section.*

## 3 VISUAL REPRESENTATIONS IN GAMES

Before choosing the style and technique to use, various possibilities were researched by looking at the history of video games.

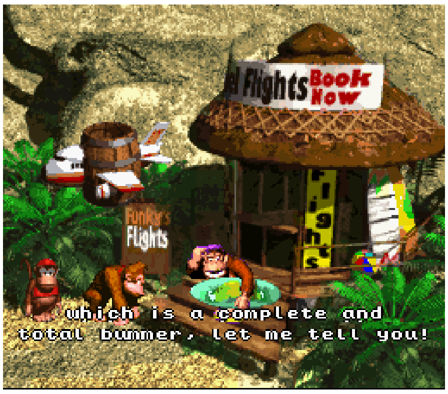
This section attempts to look at the various visual representations used by games with 2D mechanics – scrolling games in particular.

### 3.1 Pre-rendered 3D graphics

Donkey Kong Country (SNES) [P1] uses high resolution 3D computer-generated images rendered as bitmaps (Uren, Vince(Ed.), 2003, p5). Rather than being drawn as pixel art, the sprites are 3D animations. Because no actual 3D computation is performed in real-time, it runs smoothly even on the 16-bit SNES hardware.

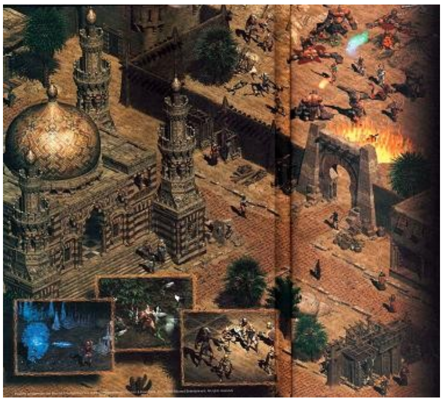
Super Mario RPG (SNES) [P2] and Diablo II (PC/Mac) [P3] use the same technique, but with an isometric view. The isometric view has no vanishing point – parallel lines are drawn parallel.

It allows map-like navigation in 4 directions. A wide range of strategy games on the PC use isometric projection. When controlling a large amount of troops in war-games, the god-like overview, which can be achieved with a distanced isometric view, facilitates game-play decisions. Another benefit is that since lines don't converge, tiling techniques may be used.



which is a complete and total bummer, let me tell you!

*[IM2] Donkey Kong Country's use of pre-rendered 3D graphics was quite distinct at its release in 1994.*



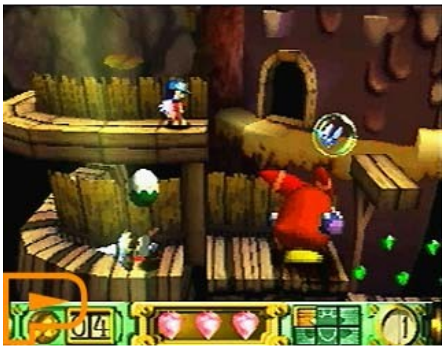
*[IM3] Diablo 2 uses an isometric view highlighting the three-dimensional characteristics of objects, and allowing two dimensional movements on a ground plane.*

### 3.2 2D Sprites in 3D environments

There are also games which combine 2D and realtime 3D graphics in their visual representation.

In the game Klonoa (PSX) [P4], the environments are 3D, but the characters are not. The first Klonoa game uses sprites for characters that always face the camera. As long as the camera is not tilted to view the characters from above, the illusion works well.

The Paper Mario (N64) [P5] series emphasize how characters are flat like paper, but this game also uses 3D environments to create depth. There is no dynamic lighting, with all the colouring done through texture painting.



*[IM4] The game Klonoa uses 3D environments. Klonoa may run around the corner, and the camera will rotate around, still facing the character sprite.*

### 3.3 3D Polygons on 2D painted backgrounds

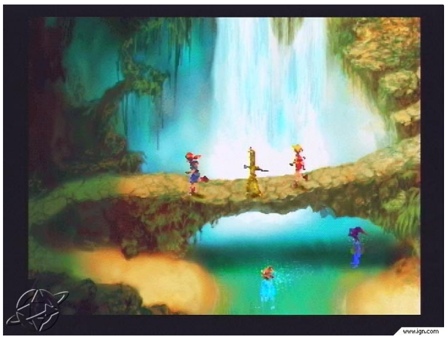
Other games do the opposite of Klonoa and Paper Mario – instead of 2D characters in a 3D environment, the backgrounds are painted and the controllable characters are polygonal 3D.

This has the benefit that the game can use several perspectives.
Since the backgrounds are unique for each area, each area has a new background painting with a new perspective. Instead of successive images for character animations, the game engine handles animations through skeletal animation or tweening of polygonal vertices.

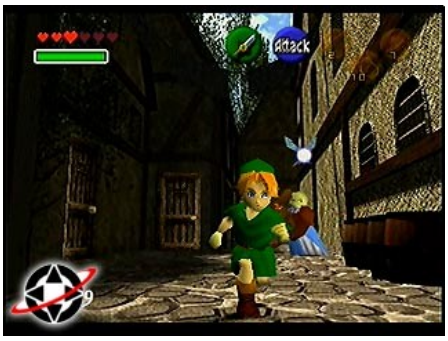
Where camera movements are required, the movements of the 2D backgrounds are usually done using scrolling techniques.

Examples in this respect are the games Escape from Monkey Island (PC/Mac) [P6] and Chrono Cross (PSX) [P7].

The Legend of Zelda, Ocarina of Time (N64) [P8] had a different way of dealing with polygon count restrictions. The game was for most purposes full 3D, but in the main square, which was lively and had a lot of polygon characters, the backgrounds were pre-rendered. In some areas, the game used a projection technique not commonly seen in other games, where the scrolling gives the effect of rotation rather than translation of the camera.



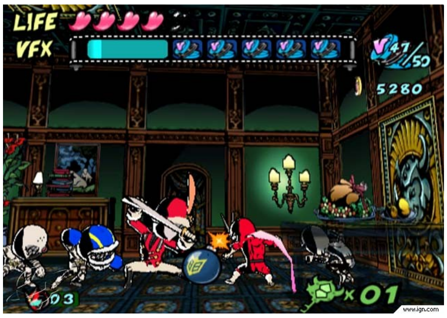
*[IM5] Chrono Cross has sections of high-detail painted backgrounds in combination with 3D polygon characters*



*[IM6] The background of this scene in the Legend of Zelda, Ocarina of Time is a bitmap image. As the character runs towards the screen, the camera will change to a different background, showing a new piece of the town. Certain background elements that need interaction, such as the doors, are real-time 3D.*

### 3.4 Full 3D graphics with restricted movement

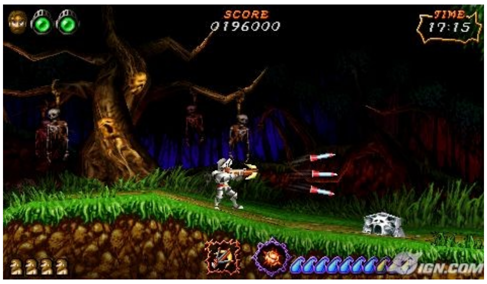
The step beyond this is to take the game to full 3D, only restricting the character or camera movement to move like a scrolling game. This can be seen in games like Viewtiful Joe (GC/PS2) [P9] and Klonoa 2 (PS2) [P10]. The art assets are three-dimensional but the games feel to high extent like a traditional side-scroller.


*[IM7] Viewtiful Joe has full 3D graphics, but gameplay mechanics are for the most part two dimensional.*

### 3.5 Variations and parallax scrolling

Another variation seen in games like Ultimate Ghosts and Goblins (PSP) [P11] and Mega Man Maverick Hunter X (PSP) [P12] is the use of 3D in the environments only on the main layer of the path and platforms. Apart from the layer the player-controllable character is walking on, the environment graphics are parallel to the camera.

Both of these games use parallax scrolling –a technique that adds depth and immersion to scrolling 2D graphics. Background elements move at slower speed than foreground elements, giving a three-dimensional effect.


*[IM8] Ultimate Ghosts 'N Goblins uses 3D selectively, giving a classic action-arcade feel.*

## 4 GRAPHICS STYLE

Our collaboration did not include discussions of graphics style, as I had already chosen to get in touch with Romina because of her style. I was looking for an artist with an optimistic, imaginative and colourful approach to art. I also desired to work with someone with strong traditional arts skills –someone who would have a visual expression which is either somewhat stylized or impressionistic.

*[IM9] Romina Denti has a style and sense of color which I admire. This is an example of her previous work. Romina is not foreign to collaborations; Design by Samuele Maggi. Colouring by Romina Denti.*

However, the technique and implementation used to create the graphics was a matter of much experimentation. We failed and wasted much time along the way. Before evaluating our various attempts, I present the implementation options which were considered during the project;

# 5 THE WORLD-BUILDING TOOL

Although 2D images or 3D models may be created through the use of graphics software –assembling several of these together in various positions with their associated properties may require additional functionality; to create the game world, a software tool is needed in order to define it.

## 5.1 Tile Editors

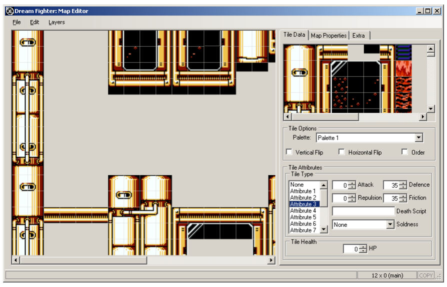
Game art started as pixel art. The amount of data displayed on the screen was limited due to severe hardware limitations, and hence the image was edited at the level of individual pixels.

Scrolls can be large areas of graphics which are continuously unravelled. Computer memory may not sufficient to store the entire scroll as unique data, and for most early games this was certainly the case.

The solution is to store pieces of unique graphics and repeat it. A method that was commonly used is tiling. The tiles are (usually) square blocks of graphics data that can be put together like a puzzle using a tile editor in a grid lay-out. The tile editor stores a tile map, with data for each cell of the grid.

Since each unique tile graphic is stored only once, and the data concerning the positioning of the tiles is stored separately, large worlds can be defined with few art assets.

Using a tile editor was investigated as a possible method to build our game world. Tile editors such as Dream Fighter Map Editor [P14] and MappyWin [P15] allow the creation of tile maps through traditional techniques. However, I found the usage of these tools to be too limiting for our purposes of visual expression. The grid-based layout is apparent in most tile-based games, and it takes great skill to hide the underlying structure. Although some tile based editors support features such as parallax scrolling, no tools were found flexible enough for our needs.



Palette: Palette 1

☐ Vertical Flip ☐ Horizontal Flip ☐ Order

0 Attack 35 Defence

0 Repulsion 35 Friction

Death Script

None Soldness

0 HP

12 x 0 (main)

*[IM10] Screenshot of Dream Fighter Map Editor*

## 5.2 Custom game editor

Due to not finding an appropriate tool for building the game world in 2D, the development of a custom tool was considered a viable option. By developing the world definition tool from scratch, the tool can be customized for the exact needs of the game.

For example, instead of a grid layout, the graphic elements could have been placed arbitrarily. Any other custom data could be attached to each element, such as transparency, tinting, depth, speed of scrolling, etc.

Two challenges were faced;
The first was time restrictions. Creating a new GUI-based application to define all world properties may have proven too time-consuming in this situation.
The second was that Romina and I use different platforms. An application compiled on Windows will not run on a Mac. This complicated matters, as rapid and easy cross-platform deployment was necessary.

Despite much thought, those were the main reasons for deciding against this option.

### 5.3 Using existing graphics tools

Would it be possible to extract the data needed for the game world through existing software or file formats?

Typical 2D graphics applications were disregarded, as they would normally not work with depth information apart from layering. We did not want to go for a completely flat look.

This left us with 3D software. It would be possible to lay out 3D objects or 2D planes in established 3D graphics tools such as Maya or 3ds Max, then export that data through any of the file formats available. This data could then be read in by the game engine, and displayed.

Since 3D software packages immediately supply the interface and application, this could provide a means of defining the game world without developing a custom application for the task.
Additionally, most 3D software is available on both Mac and Windows, solving cross-platform issues.

These arguments gave me reason to investigate this option in further detail.

A set-back was that Romina did not have any experience with 3D modelling tools. But she was interested in learning if I would provide guidance and direction.

We chose Autodesk Maya [P17] as our tool of choice, due to the fact that I had previous experience with it and could provide Romina with any support needed to use it.

### 5.4 Interchange file formats

Once the data is created in the software tool, how can it be transferred for display in the game engine?

Using an interchange file format can allow the data to be collected independently of the tool used. I had acquired some knowledge by a previous research project on the subject, where I investigated the Collada file format.

Collada (.dae) files can be exported from a wide range of 3D tools. The data is output in xml format according to a special schema. Although I had the benefit of having previously developed a tool for reading certain data from the .dae files, using Collada did not necessarily equate to the quickest and most flexible way forwards.

Usage of an interchange format like Collada may be beneficiary in other circumstances. However, I dismissed the Collada file format as a method of storing the game level data for the following reasons;

The Collada files can be outputted with the xml tags or data formatted in several ways. Thus, the application reading the files has to accommodate several scenarios.
The files may also contain a lot of redundant data not needed by the game engine.
The formatting of the data does not necessarily correspond directly to the data or interface of the host application. Sometimes certain data is needed, such as the texture files used, but it is hidden deep within a hierarchy of interlinked xml tags.
Loading these files directly would not be optimal, and file sizes of the document object model libraries add a significant amount to the application footprint.

In fact, Sony Computer Entertainment, the developer of the format, encourage on their forums that developers not load the files directly, but make some kind of filter application to output the data optimally for a specific use, such as a game engine. This of course adds another step to the workflow.

Collada is of course not the only option. File formats like obj, fbx and dxf are other possibilities. However, compared to Collada these files miss one feature that is important to game development; the output of custom game data.

In order to get any data beyond the standard visual properties of the object, export of custom data is needed.

Since our environments existed in the context of a game world, we did not want to exclude the possibility of exporting and reading custom attributes from Maya.

### 5.5 Custom file format

Having confined ourselves to Maya as a tool for defining the game world, another option would be to use the Maya API to directly access the data from Maya, and output that data to a custom file format.

The API allows access to most of Maya's internal data, and it may be output and formatted for our specific use, in one single step.

How difficult would it be to learn the Maya API to accomplish this?
It proved to be a manageable task, particularly through the use of resources such as www.robthebloke.org –where practical source code examples are provided. The standard Maya documentation was also helpful, particularly for reference.

The Maya API was the route that I eventually chose to gain the data needed for building the game world.

# 6 CHOOSING THE VISUAL REPRESENTATION

After evaluating the various techniques employed by other games, I had two proposals for Romina:

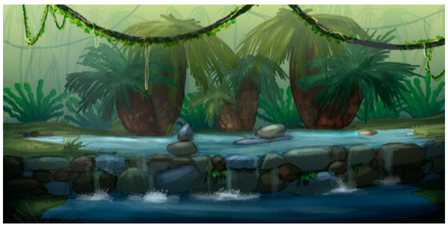
*One of Romina's original concept sketches for our project.*

1. Creating the game world through parallax scrolling
2. Creating the game world through a combination of polygon 3D and 2D planes

I considered these two techniques suitable according to the design document of the game we were preparing to build. I consulted with Romina to choose the appropriate technique.

## 6.1 Creating the game world through parallax scrolling

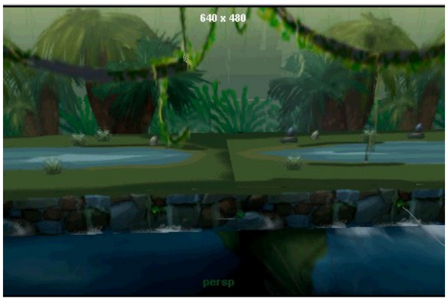
This technique would give the game a sense of depth but without using 3D modelling.
It would allow Romina to stick to painting without thinking about many of the complexities of using 3D software.
However, I considered this the "easy route" –if a talented painter like Romina could take the step to improve her work with more 3D elements, we could create an interesting visual expression with more depth.



*Another of Romina's concept sketches.*

## 6.2. Creating the game world through polygon 3D and 2D planes

I had the idea of a 3D panoramic world that would scroll from left to right, with beautiful vistas –giving the sense of travelling through landscapes with depth and scale. Surely, 3D was required, I reasoned; and with a painter to create the world, and using many sprites, the computerized polygon look would not be an issue.



*Screenshot from a video I created for Romina, where I split up her graphics to roughly illustrate the potential benefits of a 3D painting.*

However, I did suspect that getting a colourist with no previous experience of 3D software to use a new and unfamiliar technique could be challenging.

So I thought the best thing to do, was to present both techniques to the artist, with video examples of games that use both techniques. I also used some of her concept sketches and cut them into pieces to create a video giving a rough idea of the two techniques.

I expressed that I favoured the second option, but was intending to take her opinion strongly into consideration.

It turned out that Romina was willing to try to use 3D features. We would use Paper Mario 2 (GC) [P1] as a frame of reference for the amount of 2D vs. 3D to use.

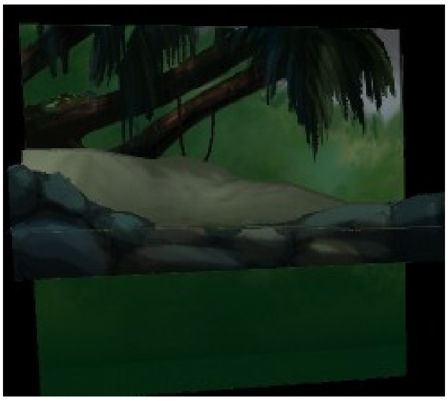
## 6.3 Learning 3D software

Our theme was an imaginative and colourful "jungle" landscape.
Romina provided several quick concept sketches for a jungle world which consisted of a warm and sunny village area, an interior temple area and a deep exterior jungle. These were visualizations for a game world we planned to develop.

The concept sketches confirmed that we shared the same vision for the game world.
The task was then to turn our ideas into graphics running in a game engine.

Romina embarked on a journey through the Maya documentation which lead to weeks passing without productive progress.

Trying to resolve the situation, I thought that if Romina would just learn the relevant Maya features, we would start making graphics quicker. I pointed Romina to relevant chapters in the Maya documentations.


*An attempt combining 3D paths with 2D painting.*

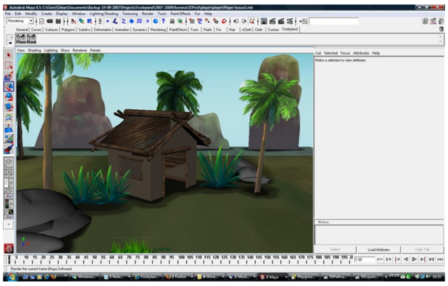
I received feedback that progress was slow. I attempted to write my own documentation, trying to strip Maya to the bare basics of features needed, and explaining the features as clearly and concisely as I could.
For further detail, see Appendix 3 – Maya How Tos.

I received positive feedback on the effort, and we started seeing some of our first 3D graphics.

## 6.4 3D Panorama

The result was not as artistic as the concept sketches. The move to 3D caused loss of the charm present in Romina's other work.

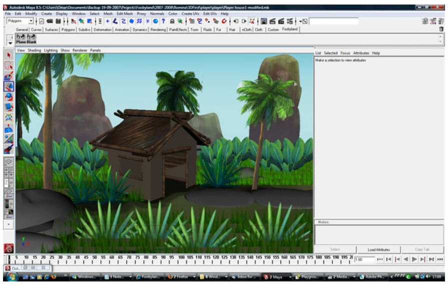
Our first reaction was one of persistence; we are just starting out, and trying something new, so let's keep going to improve the situation.


*Attempt to illustrate for Romina how shadows can be used in 3D software to add contrast to the image.*

I provided lengthy explanations of game graphics and further support. I took Romina's graphics and rebuilt them in various ways to demonstrate things that could be done with the software –e.g. shadows to create more contrast and 3D paint tools to change a texture within the scene.

Things did get better, but we realized that this was not efficient enough. Our goal was not just to output a tiny area of graphics, but create a visually stimulating larger game world, and the workflow to make that happen.

Romina felt restricted by the 3D tools. She could not paint quickly and freely. She had to move vertices and deal with all the troubleshooting of using the 3D software.



*This was our most sophisticated result using 3D environments. The image elements were painted by Romina, and I did the placement of all the trees and shadows. However, we had to give up, due to this route being too time-consuming and complicated for our purpose.*

We had to conclude that in this context, our attempt was a failure.

### 6.5 3D Parallax scrolling

Our previous experience, lead to two realizations:

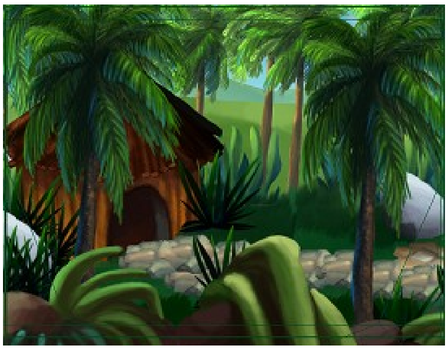
1. An efficient workflow should be intuitive and easy.
2. Parallax scrolling may be done with the full benefits of 3D.

In the case of Romina, abandoning her well-proven and experienced methods was not a good idea within such a time frame.

At this point, I thought people should play on their strengths. I contacted Romina because she had talents I did not have. How could Romina's strengths be put to better use?

The answer was as obvious to Romina as it was to me; "Paint more".

So we decided not to use any polygons at all. Everything in the game would be 2D sprites composited on top of each other. That meant all image elements would be painted in 2D software and then assembled in the 3D software as planes. It is possible to achieve depth without polygon-shapes through scale, overlap and speed differences of the image elements.



*Screenshot from the game engine. Our final solution consisted of 3D parallax scrolling, with 2D planes of painted graphics laid out in 3D space.*

This is where the idea of 3D parallax scrolling came in.
Parallax scrolling is a common technique in 2D games, but most games have just a few layers.

But what if the objects are positioned in full 3D without any layers at all?
The Z depth of the image would then give the functionality of a layer.
And with a large amount of sprites positioned at different depths in 3D space, the effect of the parallax scrolling is increased.

# 7 THE SOFTWARE SOLUTION

Three modules were needed to create our graphics workflow:

1. Maya Interface
2. Maya Exporter
3. Game Engine

The Maya Interface makes Maya more efficient to use for our specific purpose, the Maya Exporter writes the data needed to define the game world, and the Game Engine reads the data and renders it.

## 6.1 The Maya Interface

An important function needed when building a world consisting of 2D images is to add a bitmap graphic and position it in the scene.

As Maya is built for 3D graphics, this is not entirely straightforward. A polygonal plane must be created, its width, height, subdivisions may have to be defined, a material assigned, a texture node connected, and finally finding the texture file.

A one-step process would make the workflow more efficient.

We had many iterations of designing the Maya interface to suit our needs. The process can be compared to a standard software development cycle where bugs are fixed and features are added according to client requests.

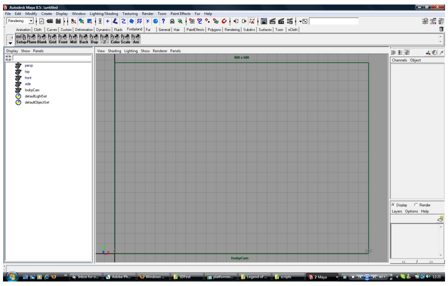
I present an overview of the most important features of the Maya Interface. Full details can be read through the documentation, Appendix 1 – Maya Interface: Usage Instructions.

The Maya Interface is implemented through a Maya shelf. The shelf includes several buttons that perform actions. The actions are implemented through MEL and Python scripts. Thus, cross-platform compatibility can be achieved.

### 6.1.1 Maya Interface Features

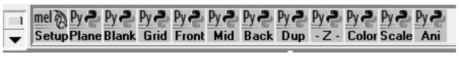
One button, the setup button, must be pressed before all others.
It sets up a suitable environment, with a custom camera that emulates the camera in the game engine, an outliner on the left for easy selection of objects, and removal of unnecessary panels and interface elements.

The plane button brings up a dialog, allowing you to select an image to insert into the scene. A plane will be created with the same dimensions as the image, and the texture assigned.



*The Maya Interface sets up a camera with an identical view to how the graphics are rendered in the game engine. The outliner to the left helps ease selection of objects. The whole interface setup is executed when pressing the "Setup" button on the shelf.*

The Z button has multiple functionalities to help create parallax scrolling:
The *Z slider* moves the object backwards and forwards in space.
The *Speed slider* moves the object back in space without changing the relative size of the object in screen space. The result is that the image size will not change, but it will scroll slower or quicker.
The *Scale slider* changes the size of the object. The Z property will remain the same.



*Closer view of the Maya Interface shelf.*

Front, Mid, Back buttons toggle the visibility of objects according to their distance from the camera.

The fit button easily sets the speed of the image scroll from one length of screen-space to another. E.g. it is possible to make a 1024 wide image scroll over a 2048 wide area. This means that the image scrolls at half speed.

The Ani button lets you quickly create an animation to scroll the camera to preview the game world. You simply set start and end point coordinates for the animation. After creating the animation, it is possible to scrub in the time line to quickly see another part of the game world.

### 6.2 The Maya Exporter

The Maya Exporter is a compiled program. It writes an ascii file with the data needed to build the game world. It writes the information in a specific format that can then be read in by the game engine. The Maya exporter also consists of a MEL file which creates the Exporter options interface. This allows setting options before export. Currently, the only option is the target path to look for textures.

The file is output as an .oes file, and contains information about the textures used in the scene and the positioning of all objects.

Creating a new level can be done in a single step. One can simply export the .oes file and the game engine will read the file, provided all the textures needed are in the texture path set in the options.

The implementation of the Maya exporter is as follows:

1. Loop through all mesh objects
2. Get position data
3. Get colour data
4. Get texture data
5. Output to file stream

#### 6.2.1 Maya Exporter – oes file details

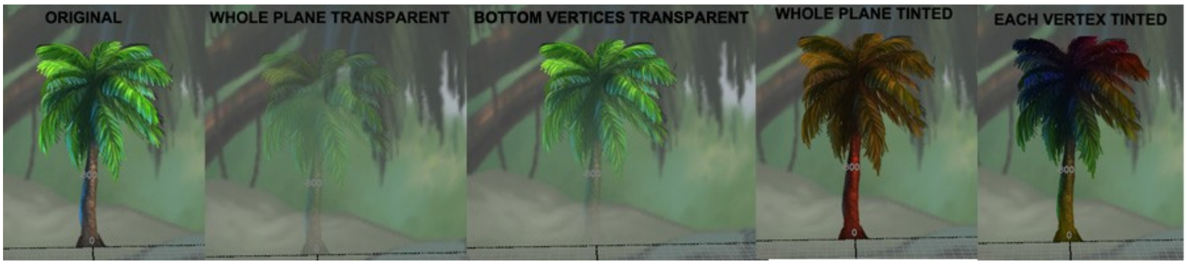
The .oes file outputted by the exporter has a simple structure.

The first line is a header telling the program how many textures and objects need to be created. This allows the memory to be allocated in advance.

Each object also has colour information so that an object may be tinted or made transparent on an individual basis. The texture index refers to one of the images in the texture list, which shall be used to display the object.

The data is output as follows:

```
<number-of-textures> <number-of-objects> <texture-path>
<texture1_filename>
<texture2_filename>
<texture3_filename>
…
<object1_x> <object1_y> <object1_z> <object1_scale> <object1_red>
<object1_green> <object1_blue> <object1_alpha> <object1_texture_index>
…
```



*Suggested graphic effects for objects in our game engine. Upon consultation with Romina, we decided that tinting and alpha for the whole object was enough.*

## 6.3 The Game Engine

The game engine uses the Playground SDK [P16] to provide a framework for drawing graphics. Application details such as the main application loop are not the focus of this report, so those details will be omitted here. Instead, I will present the drawing specific features of the engine.

All drawing operations and loading of the .oes file is located in readgamedata.cpp After reading the header, the program reads the object properties into data structures. Each object has a structure called spriteProperties which stores the same properties as in the oes file. In addition, the texture dimensions are stored in order to assist in determining if the object is within range of the screen.

The drawing operation consists of the following steps:

1. Loop through object list, convert 3D object co-ordinates to 2D screen co-ordinates.
2. Detect whether objects are within the range of the viewport. Make a list of these objects.
3. Sort the list according to z value
4. Draw the objects according to their properties

The game engine may be tested by starting:
gameEngine/skeleton/gameEngine.exe

## 8 INTERFACE IMPLEMENTATION AND DESIGN ISSUES

With the exception of the Setup button, the Maya Interface consists of Python scripts. Full implementation details may be found in the python script file in the mayaInterface folder.

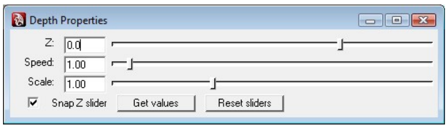
Python was used instead of MEL due to its standard libraries that perform many useful functions outside the realm of Maya. Scripts from www.python.org were used to get the dimensions of the imported image. The plane created in Maya is then sized according to the dimensions of the image. All the connections between the texture and the plane are handled so that the image is ready to be moved around immediately.

As the images are all 2D, the idea was to create a 2D interface within Maya. A perspective camera is used but rotation of the camera is locked off.
This actually differs from the game engine, which works without a perspective projection. Emulating the Maya viewport and figuring out the conversion from 3D co-ordinates to 2D sprite drawing operations was done through trial and error.

Usage patterns were examined in order to determine which operations were performed frequently.

At first, all depth was handled through a Z button that simply moved the objects backwards or forwards in space.

The most efficient use of texture data is when the texture corresponds on a 1:1 scale to the game resolution.
Moving the objects backwards caused a high resolution texture to have redundant detail, and moving an object forwards caused the texture to have a lower resolution than necessary.



Z: 0.0
Speed: 1.00
Scale: 1.00
☑ Snap Z slider

*The user interface evolved from a single slider to move objects in space, to a dialog box with better controls.*

The first iteration solved this through the use of a scale button to compensate for detail. Manually working with scale and depth was cumbersome. More sophistication was needed.

This was solved through the use of a "speed" slider which keeps the position and resolution of the object. The visual effect of changing the speed is to change the layering and speed at which the object moves across the screen when the camera is scrolling.
This is implemented so that the object would be moved backwards or forwards in space, but the scale would be automatically compensated for the change, giving the object the same dimensions in screen space.

Romina was used to work in Photoshop. Often it was desirable to take graphics from Photoshop to Maya without spending excessive amounts of time placing individual objects.
Using the "Fit" feature, layers can be laid out in Photoshop, and exported so that the effects of parallax scrolling can be seen in the game engine in a comparatively short time.

Each layer would be exported separately as a large chunk, and once in Maya, the layer can be "stretched" to scroll over a larger amount of screens. For example, a painting of mountains in the background which is 2 screens long may be "fitted" to stay there for the length of a 4 screen period of scrolling. For a visual tutorial on the feature, see Appendix 2 – Parallax Scrolling: From Maya to Photoshop.

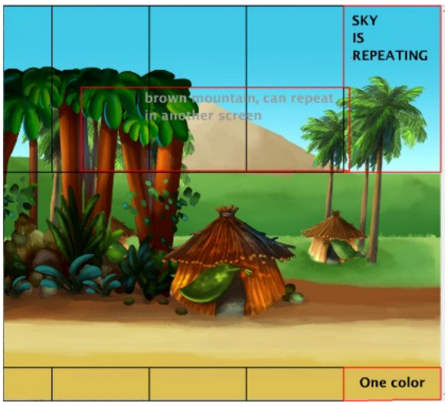
The animation button creates a keyframe animation on the camera, using attributes specified for camera start and end points and the time period to scroll. After the animation is created, it is convenient to use the time slider to scroll the game world.

## 9 CHALLENGES ENCOUNTERED

The collaboration with Romina was dynamic and iterative. We needed numerous and continuous modifications –software had to be improved, and graphics had to be adjusted due to visual problems and game-specific requirements.

The main challenge for Romina after giving up on using any form of polygon 3D in the environments, was to not treat the graphics as a painting, but to consider many of the new game specific issues.

One of these issues was the amount of image data used. We had to find an efficient way of dealing with the problem, as excessive amounts of data result in slow load times, wasteful use of memory, and sluggish performance.



*Explaining tiling techniques to Romina, even through visual means was challenging. I eventually assumed the responsibility of optimizing the game graphics, and let Romina concentrate on painting.*

Early on, I endeavoured to explain the concept of cutting images into pieces so that these pieces may be reused. However, my request lead to visual results that turned Romina's work into collage-like images, without actually reducing the amount of image data.

*Parallax scrolling and tiling was a challenge at first. Here, image elements are disjoint and scrolling at different speeds. The issues were improved through better communication and improvements to the user interface.*

We did not find a way to let Romina do this efficiently. Every time we tried, it resulted in a collage look. We settled for a middle-way, where Romina would make all her paintings easily separable in a psd file. That way I could later go over all her graphics, cut and edit it so that it can be rebuilt with higher optimization.

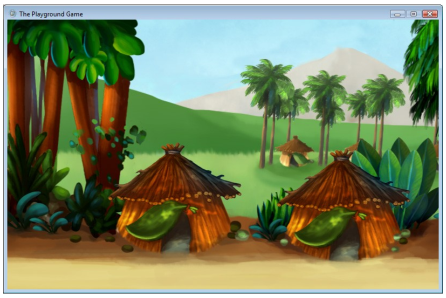
Another challenge was the importance of a clear path or platforms to jump or walk on in a side-scrolling game. The paths lacked definition and it was unclear where a character was supposed to walk.

The parallax scrolling technique was new to Romina, who created her images not fully realizing how this technique can be used to its potential. Layers of paint were also of the same size, although the amount of screen space occupied by a background element is much less than those in the foreground. Mountains in the background need not be painted for the entire scroll. Our first attempts at using parallax scrolling were not very good, but it improved with time.

In the area of software, Romina reported that it was problematic that image elements are selected as square planes –not taking into account their transparency. This makes it difficult to select objects which are behind others. Our current method to work around this problem is by using a combination of the outliner, display layers, and buttons that hide/display objects based on distance.

## 10 CONCLUSION

I have attempted to look at various techniques used by other games in order to determine what visual representation to use in our proposed game.



*The look of the final game graphics.*

Technologies available, time budgets and the skills of team members are all a factor in determining what technique is suitable for a given situation. In our case, we achieved the most efficient workflow through the use of a workflow which resembles the original artist’s previous experience as closely as possible.

Additionally, we found the use of a Maya interface and a custom exporter to be a successful solution for building and recreating the game graphics for use in our own game engine.

## BIBLIOGRAPHY

Andre Lamothe, “Tips and Tricks of the Windows Game Programming Gurus 2nd Ed.”, Sams Publishing, Indianapolis, IN, 2002

Arnie Katz, Laurie Yates, “Inside Electronic Game Design”, Prima Publishing, Rocklin, CA, 1996

Bob Bates, “Game Design: The Art and Design of Creating Games”, Prima Publishing, Rocklin, CA, 2001

John Vince (Ed.), “Handbook of Computer Animation”, Springer-Verlag, London, 2003

Mark Pilgrim, Dive Into Python, Apress, Berkeley, CA, 2004

David Parsons, “Object Oriented Programming with C++ 2nd Ed.”, Continuum, London, 1997

Stanley B. Lippman, “Essential C++”, Addison-Wesley, Reading, MA, 1999

The Encyclopaedia Britannica, Encyclopaedia Britannica Company, London, 2008

http://www.gamedev.net/reference/articles/article1269.asp Axonometric Projections - A Technical Overview, Thiadmer Riemersma (11 Feb 2008)

www.robthebloke.org, The Maya Exporter Fact-file, Rob Bateman, (11 Feb 2008)

www.collada.org/forum Public Forum, Sony Computer Entertainment Inc., (11 Mar 2008)

## PRODUCT REFERENCES

[P1] – Donkey Kong Country – SNES – Rare
[P2] – Super Mario RPG – N64 – Nintendo
[P3] – Diablo II – PC/Mac – Blizzard Entertainment
[P4] – Klonoa : Door to Phantomile – PSX – Namco
[P5] – Paper Mario – N64 – Nintendo
[P6] – Escape from Monkey Island – PC/Mac – LucasArts
[P7] – Chrono Cross – PS2 – Squaresoft
[P8] – The Legend of Zelda: Ocarina of Time – N64 – Nintendo
[P9] – Viewtiful Joe – GC/PS2 – Capcom
[P10] – Klonoa 2: Lunatea's Veil – PS2 – Namco
[P11] – Ultimate Ghosts ‘N Goblins – PSP – Capcom
[P12] – Mega Man Maverick Hunter X – PSP – Capcom
[P13] – Paper Mario 2: The Thousand Year Door – GC – Nintendo

[P14] – Dream Fighter Map Editor – http://snesdev.net/
[P15] – Mappy Editor - http://www.tilemap.co.uk/
[P16] – Playground SDK – Playfirst Inc. – http://developer.playfirst.com
[P17] – Autodesk Maya – http://www.autodesk.com/maya

## IMAGE REFERENCES

[IM1] - Taiwan National Palace Museum – http://www.npm.gov.tw/
[IM2] – Omar El Sawi, screen capture
[IM3-8] – www.ign.com
[IM9] – Romina Denti
[IM10] – Marcus Rowe, screen capture

## APPENDICES

Appendix 1 – Maya Interface: Usage Instructions
Appendix 2 – Parallax Scrolling: From Photoshop to Maya
Appendix 3 – Maya How To’s

## APPENDIX 1 - Maya Interface: Usage Instructions

### Installation instructions

1. Restore Maya to default settings:
    a. Go to ~/Library/Preferences/Autodesk/maya/8.5/prefs/
    b. Delete userPrefs.mel

2. Add custom scripts:
    a. Find your scripts folder. It should be something like:

        ~/Library/Preferences/Autodesk/maya/scripts OR

        ~/Library/Preferences/Autodesk/maya/<version>/scripts

    b. Paste oesSetup.mel and oesScript.py here.
3. Add shelf buttons:
    a. Find your shelves folder:

        ~/Library/Preferences/Autodesk/maya/<version>/prefs/shelves

    b. Paste shelf_oesExporter.mel here.
4. Start Maya. Find the oesExporter shelf tab.

***Note:***

Pressing the Setup button will change the Maya layout and interface. To Restore, go to Window → Settings/Preferences → Preferences → Edit → Restore Default Settings → Save.

### Usage instructions

There will be a new collection of buttons:

***Setup:***
The other buttons will not work before clicking this button first.
It will create a new layout. On the left is the outliner: you can select all objects in the scene in a list. All planes will have the name of their texture, prefixed by plane__.
On the right is a frame which shows you the game screen -which is 800 x 600 pixels.
Hold down Alt + middle mouse button to pan the screen.
The grid allows you to position objects precisely by holding down x when you move the object.

***Plane:***
Allows you to insert a new image in the scene. The plane will have the same dimensions as the original image. The move tool will be activated automatically so you can position the object.

***Blank:***
Allows you to insert a plane without a texture. You can then right-click it to assign one of the existing textures. You should not normally need to use this button, as duplicating is much quicker.

***Grid:***
Turns the display of the grid on and off.

***Front:***
Turns on/off the display of all objects that are in the foreground.
This is useful if they are obscuring the view and you want to see the others more clearly.
Foreground objects are any objects closer than 150 in Z depth.

***Mid:***
Turns on/off the display of all objects that are at a mid-range depth.

***Back:***
Turns on/off the display of all objects that are in the background.
Background objects are any objects further away than -150 in Z depth.

***Dup:***
Click to make a copy of your object.

***Z:***
Brings up the Depth Properties dialog:
*Z* to move the object back in space.
*Speed* to move the object back in space *without changing the relative size of the object* in screen space. The result is that the image size will not change, but it will scroll slower/quicker.
*Scale* to change size of the object. Z will remain the same.

*Snap Z slider* will snap the Z slider to whole numbers.
*Get values* will get the Z, Speed and Scale of the currently selected object so it doesn't "jump".
*Reset sliders* will make the sliders look the same as they did when you first started the setup, and give them the default values.

***Color:***
Brings up a dialog where you can select a color and transparency. Click "Apply" to change the color and/or transparency on the selected object(s). This can be useful for example to create color variation without changing the texture.

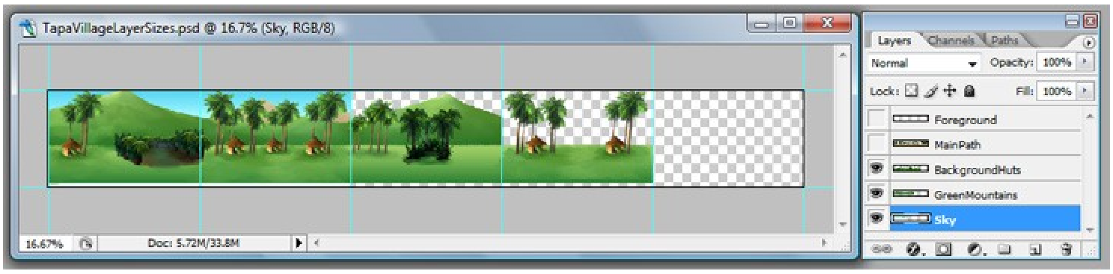
***Ani:***
Lets you preview the scene with scrolling animation.
Press "Play" to have a "tour" of your world.
Press "Stop" to continue working.
Pressing Play and then Stop will reset your camera to the default position.
Change the default numbers for the start and end points to customize the scrolling to the size of your world.

# APPENDIX 2 - PARALLAX SCROLLING

*From Photoshop to Maya*

In Photoshop –when you work:

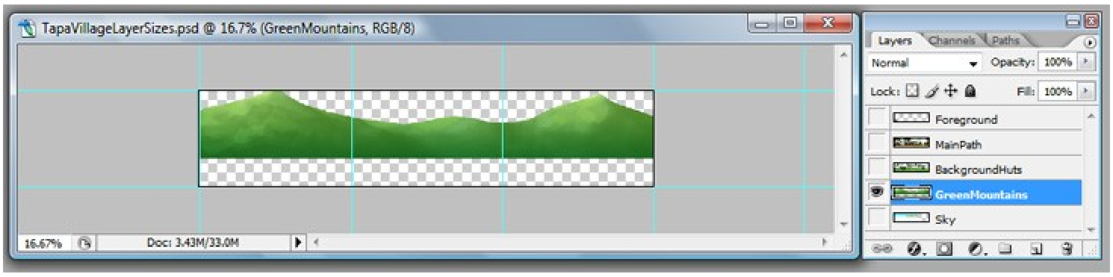
1. Lay out number of screens
2. Paint background screens shorter than foreground screens
3. Put each speed of parallax on a separate layer.



Opacity: 100%
Fill: 100%

In Photoshop –to export to Maya:

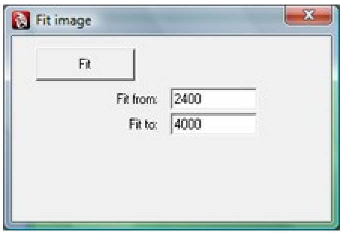
1. Choose one layer. Disable visibility of other layers
2. Crop the canvas to the size of the layer you are exporting:
   a. Image →Canvas Size
   b. Input the size:
      For example if it is 3 screens wide: 3 x 800 = 2400
   c. For Anchor, choose left ← button
   d. File → Save As
   e. Go back one step in history to restore original canvas size
   f. Repeat steps for other layers



Opacity: 100%
Fill: 100%

In Maya:

1. After clicking Setup button, click Plane
2. Choose the image. It will be positioned at normal size and scrolling speed
3. To make the image scroll-speed fit a specific length of screens, use *Fit button:*
   a. In *Fit from* field, input the original size of the image.
   b. In *Fit to* field, input the screen space you want the image to scroll for.
4. For example, for the 3-screen mountain layer to scroll for 5 screens. Input:
   a. Fit from: 3 x 800 = *2400*
   b. Fit to: 5 x 800 = *4000*



Fit from: 2400
Fit to: 4000

# APPENDIX 3 - MAYA HOW TOs

-------------------

Create a plane:

-------------------

1. Go Create -> Polygon Primitives -> Plane
2. Drag to create a flat plane

------------------

Selecting

------------------

1. If the object is not already selected: Press q for select tool OR press the button in the tool box on the left side of the screen. Left-click the object.

(Another way to select an object: There is a field on the top-right of the screen. If you gave your object a specific name, type the name here, and press enter).

---------------

Change properties

-----------------

1. On the right of the screen you can see the properties of the object. There are two ways to view the properties: "Channel Box" or "Attribute Editor". Channel box is quick and simple, Attribute editor has more detailed options. For texturing you always use Attribute Editor. To change between them press Ctrl-A, or click the buttons at the top.

2. (in Channel Box) To change the name of the object: Click pPlane1 (or whatever the object is called) and input a new name. For our game, certain objects may require the name to start with special letters or words, so they have a special function in the game.

3. You can give precise values for translation, rotation and scale by inputing the values in the Channel Box.

4. Find where it says "INPUTS". If you click there, you can edit the properties of how the object is created. For example, "Subdivisions Height" and "Subdivision Width", which will split the flat plane in many pieces. That could be handy if you want to model it into something else by moving around the vertices.
For flat planes with just an image on it, we don't need any subdivisions, so it should just be "1".

------------------------------

Positioning the object

-------------------------------

1. Select the tool you want:
   A. Press w for move tool, e for rotate, r for scale OR
   B. Press the buttons in the tool box on the left side of the screen
2. To change all dimensions:
   Drag middle mouse button anywhere in the viewport.
3. To change one dimension:
   Left-click one of the coloured handles, then middle-mouse drag.
4. To move in steps along the grid:
   Hold down x when moving the object. This works for example when you have selected the x dimension (red arrow on move tool), then hold down x and drag. You will move it along the points on the grid.
5. To change the pivot point:
   The pivot point is the center point of the object. When you rotate the object or snap it

along the grid, this point is used. To change where this point is in relation to the object, (1) press "Insert". (2) Modify the position of the object. Then (3) press "Insert" again to return to editing the object.

## Modifying vertices

To modify the shape of the object, you need to modify the vertices which are all the points it is made of. To do that, you have to be in component mode. Press F8 to go into component mode, or press the button on the status line at the top of the screen. Press F8 or press the button again to return to selecting objects.
Another way to start editing vertices is to right-click the object and then click "Vertex".

Select the vertices as any object, and move, rotate and scale them around.

## Modifying faces, edges, etc.

When you are in component mode, you can click some of the buttons at the top of the screen to change between face, edge, vertex, uv, etc. You can also right click the object and click the specific component you want to modify.
Vertexes are connected by edges, so by selecting edge you can modify two vertices at a time.
Edges are connected by faces, so by selecting face, you can modify several edges.
Certain modelling tools only work when you have the right type of component selected.

## Extruding

One of the most handy way of creating an object is by extruding.
1. Select a face or an edge
2. from the Polygon menu (press F3) : Edit Mesh -> Extrude
3. Use the tool

## Other modelling

Two useful tools:
"Merge" for joining two vertices together.
"Split polygon tool" to cut a polygon in pieces. Make sure that when you use the split polygon tool to cut at vertices, not in the middle of edges. Maya will allow you to cut anywhere, but it creates ugly models that won't work in a game engine. The game engine only uses triangles. Quads (squares) are allowed if all four vertices lie on the same plane.

## Apply a material

In order to change the appearance of the object, you create a material.
To create a material:

A.
Right click the object
Assign New Material -> Lambert OR

B.
Change to Rendering menu by pressing F6 OR: by selecting "Rendering" from the scroll-list on the status line at the top of the screen.
Go Lighting/Shading -> Assign New Material -> Lambert on the menu.

The Attribute editor will pop up with the properties of the material.

If you want to get back to the material properties of the object later:

A. 1. Select the object
2. Press Ctrl+a to get the attribute editor. Find the material tab and click it. OR

B. 1. Right click the object. Select "Material Attributes".

You should always use Lambert, and the only two properties you will need to change are "Color" and "Transparency". All the Checker-buttons allow you to apply a "texture". That way you can give your plane color and/or transparency according to the image file that you specify.

-------------------------------------
To map an already existing material
-------------------------------------

A. 1. Right click the object.
2. Assign Existing Material -> YourMaterial OR

B. 1. Using the rendering menu (press F6 . Polygon menu is F3):
2. Lighting/Shading -> Assign existing Material -> Your Material.
Method B works if you have lots of objects selected: you can apply a material to them all at once. Method A only works with one object. So for several objects, use method B.

---------------------------------------
To map a texture
---------------------------------------

1. Click the checker-button beside "color". (having selected the material).
2. Choose "File"
3. Click on the folder button to find your image.

---------------------------------
Turn on texturing in the viewport:
---------------------------------

Your texture will not be visible. You have to turn on hardware texturing to see it:
A. Press 6 for hardware texturing OR
B. 1. Go to the "Shading" in the viewport menu.
2. If "Smooth Shade All" is not on, click it.
3. If "Hardware texturing" is not on, click it,

Press 4,5,6,7 to switch between: Wireframe, Shaded, Textured, Textured+Lit
You can also turn it on from the viewport menu.

----------------------------------
Planar Texture mapping
----------------------------------

For 2-D images that are placed in the scene, you do need to modify the texture position.
But for any object which has a more complex shape, you need to change the position of the texture on the object.
The easiest way is "Planar mapping". If your texture is supposed to be wrapped "flatly" like on the walls of a house, this is all you need.

To planar map:
1. Change to Polygon menu (F3).
2. Select the object if it is not selected.
3. From the main menu: Create UVs -> Planar Mapping
4. You probably need to change the options, so press the little box beside Planar Mapping.
5. Choose X, Y, or Z-axis for the angle which you want to project it from. Usually use: "Bounding Box".
6. Look at the x,y,z at the bottom-left of the viewport to help you choose the right axis. Click "Apply" if u want to keep the options box on the screen. Otherwise, click "Project". If you made a mistake, press "z", and do the projection again.
7. There will be a manipulator in the viewport now.

8. Click the boxes and drag for changing the scale of the texture,
9. Click the red lines and drag to change the position.
10. Click the red cross at the bottom-left of the manipulator to modify the texture using a tool similar to the normal move, rotate, scale tools.
11. If you click accidentially deselected the objected, press z to get back to the manipulator again.
12. When you are finished, click anywhere. You might be in "component mode". Press F8 to go back to modifying objects.
13. To get back to modifying the texture placement later using the same tool, you can do a projection again OR you can find the "polyPlanarProj" in the "Inputs" in the Channel Box: Click it, and make sure you have "Show manipulator tool" selected (press y).

------------------------------
UV Texture Editor
--------------------------------

Access the UV texture editor, by going "Window" -> UV Texture Editor in the main menu.
Each vertex on an object has a "UV". UV is basically a position within the flat texture.
So for each vertex, you can change the uv: =change which exact location within your texture will be visible at that vertex.

1. Bring up the UV Texture Editor
2, A. Right Click the object, select UV
B. Go into component mode (F8). Select UV as the component to modify (click button at top of screen).
3. In the viewport OR in the UV Texture Editor select the UVs you want.
4. Use the move, rotate or scale tool to move them around in the texture editor.
5. Use ALT+Button to navigate in the texture editor as you do in the viewport.
6. Look at the menus for more operations to perform on the uvs.