Fall 2011

GENERATION OF FORESTS ON TERRAIN WITH DYNAMIC LIGHTING AND SHADOWING

Jonathan Ben-David
San Jose State University

GENERATION OF FORESTS ON TERRAIN WITH DYNAMIC LIGHTING AND SHADOWING

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

By

Jonathan Ben - David

December 2011

The Designated Writing Project Committee Approves the

Writing Project Titled

GENERATION OF FORESTS ON TERRAIN WITH

DYNAMIC LIGHTING AND SHADOWING

By

Jonathan Ben - David

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

December 2011

Dr. Soon Tee Teoh	Department of Computer Science
Dr. Chris Pollett	Department of Computer Science
Professor Roger Alperin	Department of Department of Mathematics

ABSTRACT

GENERATION OF FORESTS ON TERRAIN WITH

DYNAMIC LIGHTING AND SHADOWING

By Jonathan Ben – David

The purpose of this research project is to exhibit an efficient method of creating dynamic lighting and shadowing for the generation of forests on terrain. In this research project, I use textures which contain images of trees from a bird's eye view in order to create a high scale forest. Furthermore, by manipulating the transparency and color of the textures according to the algorithmic calculations of light and shadow on terrain, I provide the functionality of dynamic lighting and shadowing. Finally, by analyzing the OpenGL pipeline, I design my code in order to allow efficient rendering of the forest.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Soon Tee Teoh for his guidance and his support through my research process. Moreover, I would like to thank my committee members Dr. Chris Pollett and Professor Roger Alperin for their critique and support. Finally, I want to thank my parents and my dear friend Madeline Marusarz for their moral support.

TABLE OF CONTENTS

1. INTRODUCTION 8

1.1. Project Description 8

1.2. Project Example 9

2. TECHNICAL BACKGROUND 12

2.1. C++ 12

2.2. OpenGL.....12

2.3. OpenGL - Texture mapping 13

2.4. OpenGL - Lights and Shadows 13

3. RELATED WORK IN THE FIELD 15

3.1. HOW TO RENDER A FOREST IN 3D [4] 15

3.2. RENDERING FOREST SCENES IN REAL - TIME [5] 15

3.3. RENDERING OF FOREST SCENES [6] 17

3.4. MULTIRESOLUTION FOLIAGE FOR FOREST RENDERING [7] 17

3.5. VEGETATION FOR TERRAIN GRAOHICAL MODELING USING L-SYSTEMS OVERVIEW [8] 18

4. DESIGN AND IMPLEMENTATION 20

4.1. Generation of Forests on Terrain 20

4.2. Dynamic Lighting and Shadowing 25

4.3. Code Structure for rendering efficiency..... 32

5. CONCLUSION AND FUTURE WORK 33

6. REFERENCES 35

TABLE OF FIGURES

Figure 1 2D textures....9
Figure 2 Forest image with light source towards camera....10
Figure 3 Forest image with light source above....10
Figure 4 Forest image with light and shadow....11
Figure 5 OpenGL Rendering Pipeline [2]....13
Figure 6 Pyramid of 2D texture sets [5]....16
Figure 7 Regular texcell and silhouette texcell [5]....16
Figure 8 An example of cross billboards [6]....17
Figure 9 A simple L-System example [8]....18
Figure 10 Terrain-net image....20
Figure 11 An example of texture coordinates....22
Figure 12 Layers of textures in 3D trees....24
Figure 13 Paths of the sun....28
Figure 14 Paths of sun and direction of the rays....29
Figure 15 Trees between input tree and sun....31

1. INTRODUCTION

1.1. Project Description

The purpose of this research project is to generate forests on terrain with dynamic lighting and shadowing in an efficient manner. The research project is constructed in three main parts. The first part generates a high scale forest from a set of textures that has been generated from a 3D volume dataset. The second part implements dynamic lighting and shadowing in an efficient manner while taking into consideration the elevation of the terrain. The third part analyzes the code regarding the OpenGL Rendering Pipeline, in order to improve the display of the generated forest.

The 2D texture I use to generate the forest is based on a 3D volume dataset, which contains images of several trees. By arranging the 2D textures using several different methods, I create a 3D forest with several different 2D textures instead of rendering 3D datasets. This significantly improves the efficiency of the forest rendering by rendering less data. The methods which have been used on the 2D texture are: rendering specific parts from the 2D textures, changing the color, changing the transparency and the location of the texture according to the camera position.

In order to create the effects of light and shadow on the terrain I manipulate the color of textures. The manipulation is done according to the location of the light source. For example, at noon, when the sun is directly above the forest and no shadow is being created or when the sun is risen at an angle and creates its respective shadow. In the last part, by analyzing the OpenGL pipeline I am modifying the source code. By doing this, I am improving the efficiency of rendering the forest scene in each frame.

1.2. Project Example

The following textures are the textures which construct the forests. There are 32 different textures in total which are generated from the 3D dataset.

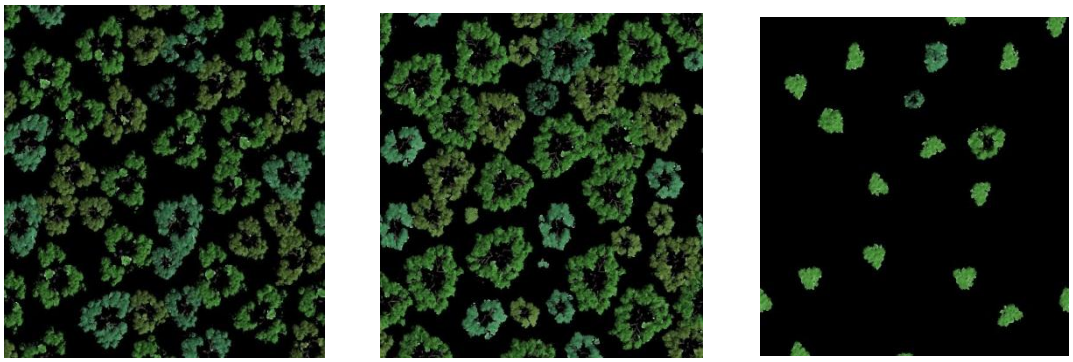


Figure 1 - From left to right: nuts_aper_0007.tga, nuts_aper_0016.tga, nuts_aper_0028.tga

Each image represents a different layer from the 3D dataset. As the number of the image decreases, it represents a lower layer of the 3D dataset. By operating several functions on the 2D textures and arranging them in layers, with changing variables, such as the number of textures, transparency and color, I create a 3D forest. For example, the function, "void locate_tree1(int k1, int z1, int h1)" allows for the isolation of one specific tree from the entire 2D texture. This has been done by defining the specific texture coordinates.

The following images are generations of forests on terrain with lighting and shadowing, which have been rendered from several layers of texture. The lighting and shadowing are created by calculating the position of the light source. Figure 2 and Figure 3 are images of the rendered forest scene from the same camera position, with different light source positions. In Figure 2 the location of the light source is angled towards the position of the camera, casting a shadow on the foremost side of the trees. In Figure 3 the location of the light source is directly above the forest, and as a result there is no shadow. Figure 4 is an image that exhibits the shadows created by the varying elevation of the terrain.

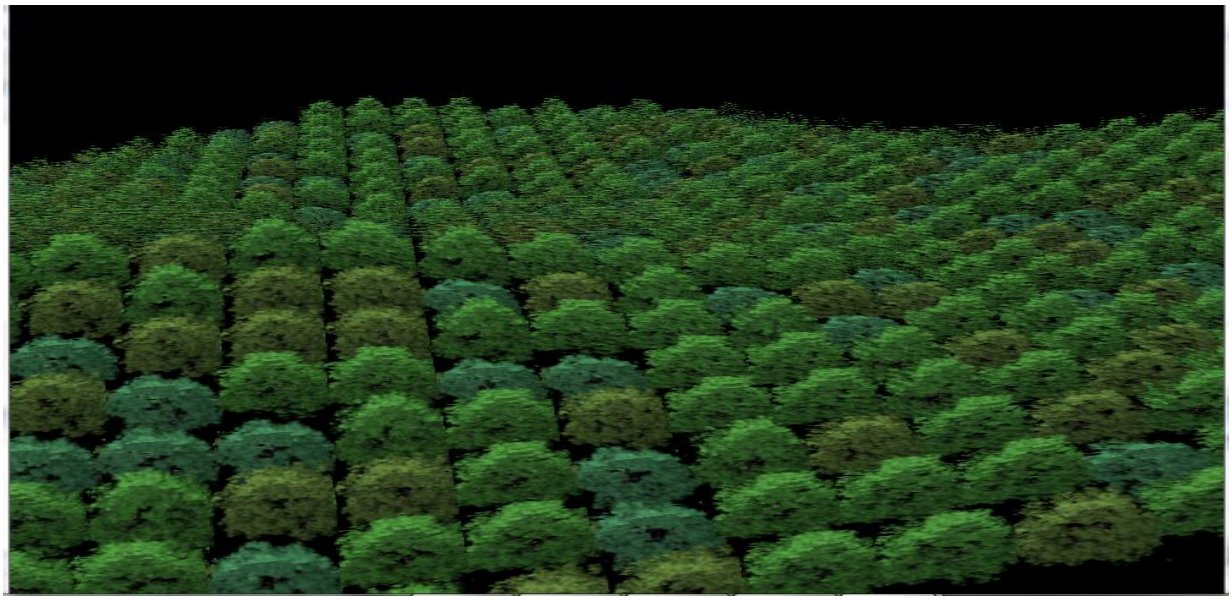


Figure 2 – The location of the light source is angled towards the position of the camera.

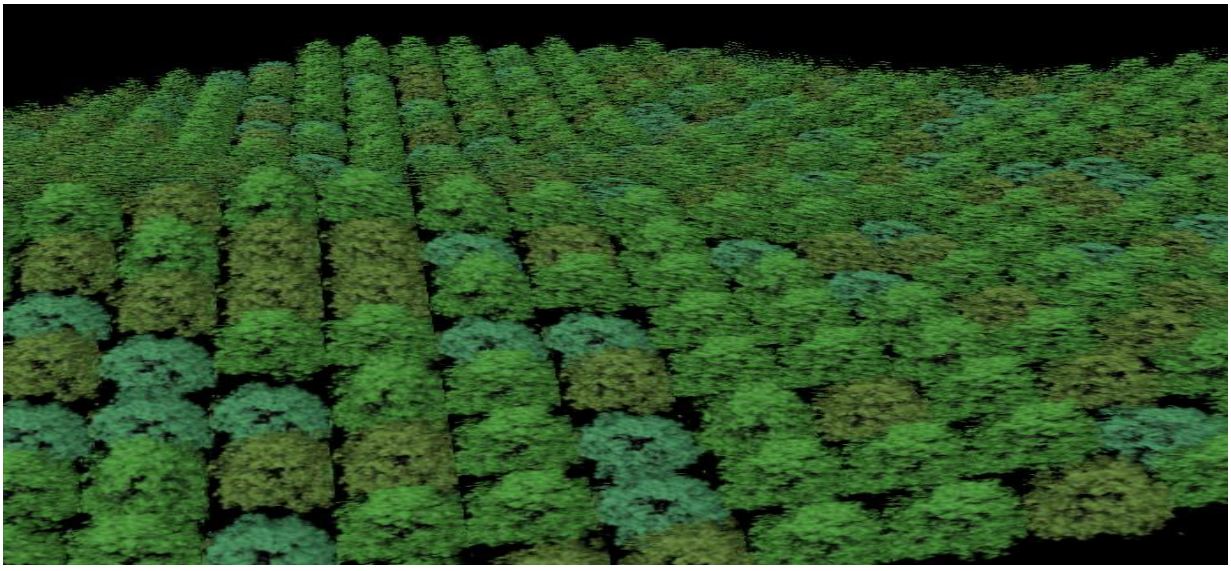


Figure 3 – The location of the light source is directly above the forest.

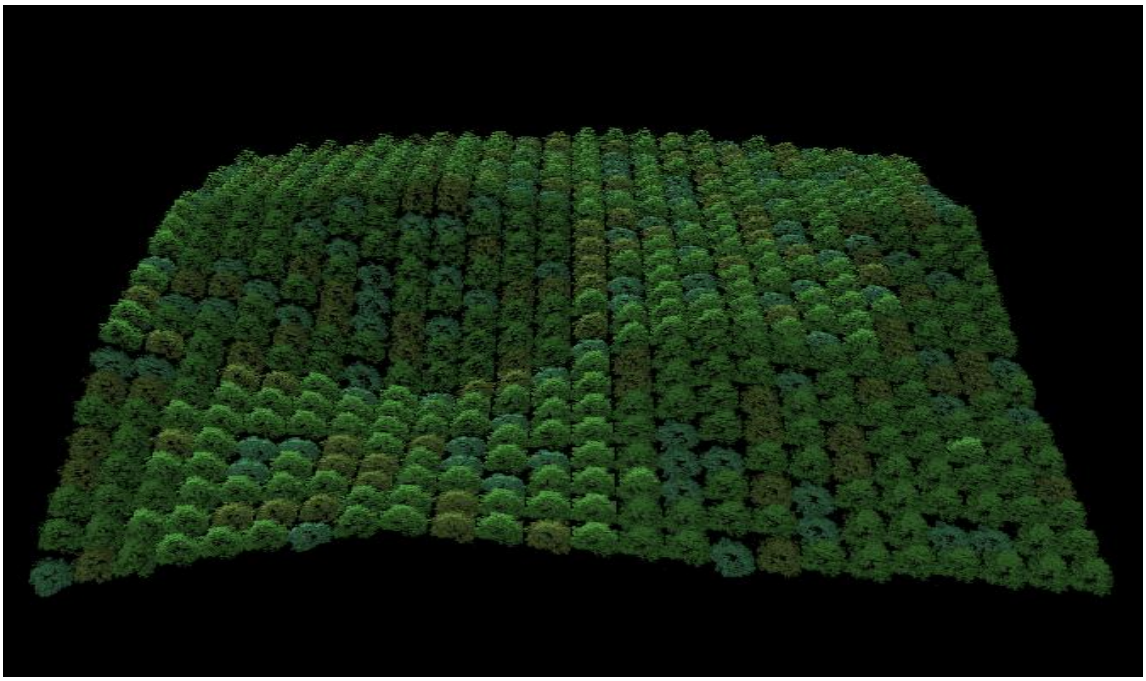


Figure 4 – This image exhibits the shadows created by the varying elevation of the terrain; the location of the light source is angled towards the position of the camera.

The structure of the code and the order of the rendering calls, such as rendering textures, have a direct impact on the rendering efficiency of the scene. An example of this would be, rendering a texture into a scene, which has been used for different objects in the scene. There are two methods to approach this. First, operate all the relevant functions related to the different objects which use the specific texture. Second, upload the relevant textures for each object. The first method is the more efficient one because each texture has been uploaded once instead of each texture being uploaded severely times.

2. TECHNICAL BACKGROUND

2.1. C++

C++ in one of the main programing languages, which is used in different applications such as: application software, device drivers, embedded software, video games. C++ is constructed from two parts, those being core language and C++ Standard Library [1]. The C++ Standard Library is constructed from C standard library with modifications and Standard Template Library. The Standard Template Library contains vectors, lists, maps, sets, templates and etc.

C++ is a language based on C language, which allows the users to use object – oriented programming features. The main features are: classes, abstraction, encapsulation, inheritance, and polymorphism. Encapsulation allows the user to define the level of access of the different object's fields and functions. The members of the class can be declared as public, protected or private. Inheritance allows classes to acquire features of other classes. By using encapsulation, the user can define the level of access of the derived classes from the base class. Polymorphism allows for creating functions or objects in different forms. There are two types of Polymorphism in C++, compile time polymorphism and runtime polymorphism. Function overloading is used for compile time polymorphism, while a virtual function is used for runtime polymorphism.

C++ templates allow the concept of generic programming. Class template created in order to allow different class to use the same functionality without repeating the entire code.

2.2. OpenGL

OpenGL is software which produces computer graphics of 2D and 3D. It has been used in different fields of computer science, such as: scientific visualization, virtual reality, video games and etc. OpenGL pipeline converts different geometric shapes and textures into pixels. OpenGL Pipeline in constructed from two main paths: the Geometry Path and the Image Path. The Geometry Path handles the Vertex Operation while the Image Path handles Pixel Transfer Operation. The main concept for Vertex Operation is that the vertex coordinates transform form object coordinates to eye coordinates [2].

OpenGL allows the user to store data in a group of arrays. The type of data that can be stores in these arrays are: vertex data, texture coordinates, color ant etc. Furthermore, the user can store commands that have been compiled by using Display list. This feature, allows for processing of the set of commands once, and it can be reused repeatedly which improves the efficiency of drawing each frame. Texture Memory is responsible for storing texture images, which is used for different geometric objects. The last process, the Fragment Operation, is responsible for converting fragments to pixels onto a frame buffer [2].

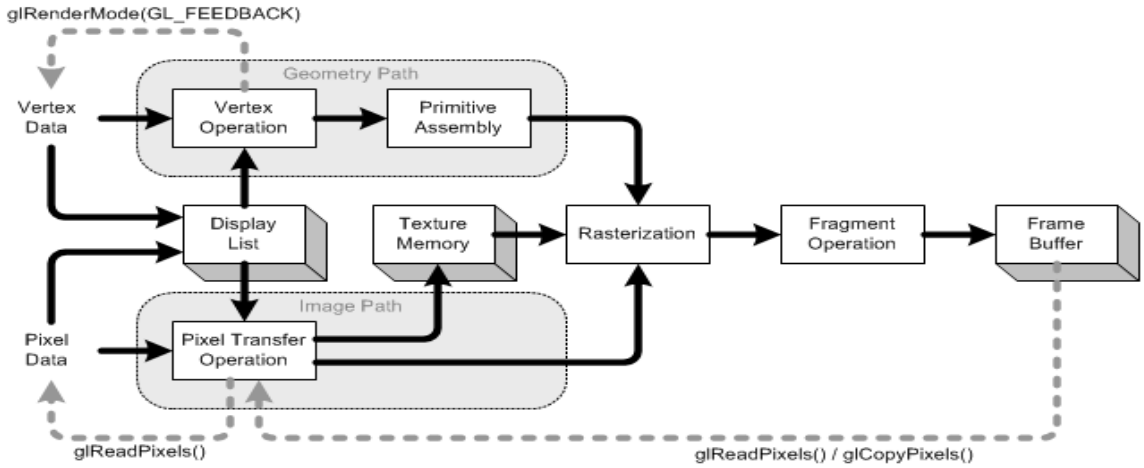


Figure 5 - OpenGL Rendering Pipeline. [2]

2.3. OpenGL - Texture mapping

Texture mapping is a feature which allows software developers to add bitmaps and images to computer graphic objects in 2D and 3D. Characteristics such as color, transparency, and brightness affect the appearance of each texture. By using Texture mapping in the different objects, the scenes and objects look more realistic. For example: In computer gaming programming and computer simulators programming there is a lot of use for Texture mapping.

The mapping between the Textures to the computer graphic objects operates by mapping the coordinates of the textures to the coordinates of the polygons that constructed the object. In order to change the shape and position that the Texture displays, the mapping ordering needs to change. Textures are mapped to a 3D world and they are rendered according to the 3D projection. Furthermore, when there is a use for more than one texture on a single fragment it is called Multitexturing. Characteristics such as color, transparency, and brightness affect the appearance of each texture.

2.4. OpenGL - Lights and Shadows

Using lighting and shadowing in computer graphics makes the rendered scene more realistic. Furthermore, in a 3D scene it contributes to the visual depth of the scene and the visual shape of different objects in the scene. The main model for illumination and shading in the computer graphics world is the Phong lighting model.

The main advantage of the Phong lighting model is to calculate the light that is being reflected off of different surfaces. Furthermore, in this model, light sources are modeled as a point light source [3], and the color components that are used are red, green and blue. In the model there are several types of light, reflection and material properties

of the surface [3]. The types of reflection are Diffuse Reflection and Specular Reflection. Diffuse Reflection reflects the light in an even way to all directions, while Specular Reflection reflects the light at an angle, which creates shiny surfaces. The types of light are: Specular Light, Diffuse Light and Ambient Light. Specular Light and Diffuse Light are lights, which will be reflected specularly and diffusely respectively. Furthermore, Ambient Light is light that is coming towards an object from all directions. In conclusion, the material properties of the surface affect the light that is rendered on the surface [3].

3. RELATED WORK IN THE FIELD

Rendering forests and plants is a very challenging subject in the field of computer graphics. There are several main factors that need to be taken under consideration while rendering forests and plants: the scale of the forest, the amount of details that the trees and plants contain, and whether the rendered scene is zoomed in or zoomed out. These factors influence the lighting and shadowing and the Real-Time rendering.

The following papers are related to different issues as previously mentioned.

3.1. HOW TO RENDER A FOREST IN 3D [4]

Hans Häggström details methods of rendering forests by using textures which surround the camera. By using textures of trees in the rendered scene, it reduces the amount of trees that need to be drawn. The main concept of the method is to render several layers of textures. The closer the layer is to the camera it contains more detail about the trees, such as trunks and leaves. In order to get a realistic forest, several horizontal bitmap layers need to be rendered in order to get the depth effect. Furthermore, the bitmaps that are closer to the camera are more transparent, which contribute to the depth effect.

In order to create more realistic forests, we need to take under consideration that in nature there are different types of forests according to the climate and terrain. The type of forest can be modified according to the type of bitmaps we use. Moreover, as the camera changes its position, the existing layers of bitmaps need to fade according to the distance from the camera, and new ones need to be created.

The paper suggests ideas for future work in order to create more realistic forests. The main idea is to create a dynamic tree model. Different variables represent the characteristics of the tree such as: the age of the tree [4], the time of the year [4] etc. Moreover, the tree model could control the level of details which the tree will render. A dynamic tree model, which possesses these features, will allow the users to define different types of forests. By changing the type and structure of the trees that are rendered, it will allow for more realistic rendering of forests.

3.2. RENDERING FOREST SCENES IN REAL - TIME [5]

Philippe Deaudin and Fabrice Neyret detail methods for rendering dense forests in real-time [5]. The method is constructed in three main parts, creating the volume data [5], creating the texture set which includes regular texcells and silhouette texcells [5], and aperiodic tilling [5]. Emphasis is placed on the methods of creating the regular texcells and silhouette texcells.

The 2D textures that are used in the paper are based on a 3D volume dataset. By rendering a set of 2D textures, the rendering becomes more efficient as a result of

rendering less data to the scene. The paper represents two slicing methods for creating a 2D texture from the 3D volume dataset. One slicing method operates in a way which slices parallel to the terrain [5], while the other one slices nearly facing the viewer [5].

The regular texcells are based on the 2D textures that are created by the first slicing method and slices parallel to the terrain. The regular texcells contains layers of 2D textures, which creates the 3D forest. Furthermore, the level of the forest's detail that is rendered depends on the number, the size and the gap between the 2D textures that have been rendered. The following concept represents the level of detail: in the finest level [5], the numbers of slices that have been rendered are 2^N, and the sizes of the textures are (L x L) [5]. For example, in level i there are 2^i, and the size of the textures are (L / (2^{N-i})) x (L / (2^{N-i})) [5].

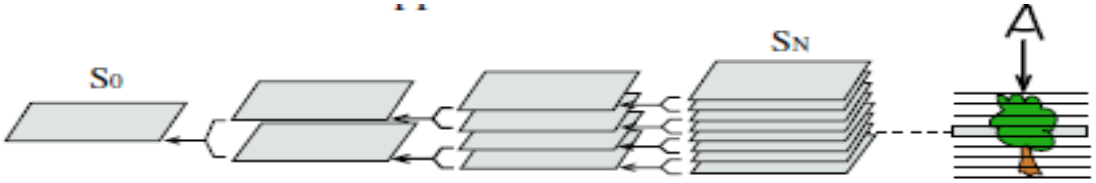


Figure 6 - Right: slicing real geometry using clipping planes. Left: Pyramid of 2D texture sets Si: i = 0...n. [5]

The Silhouette texcells [5] are created based on the second slicing method. This slicing method slices the 3D dataset at varying tilted angles. The Silhouette texcells are used to avoid the possibility of creating visible gaps between parallel slices at certain camera positions.

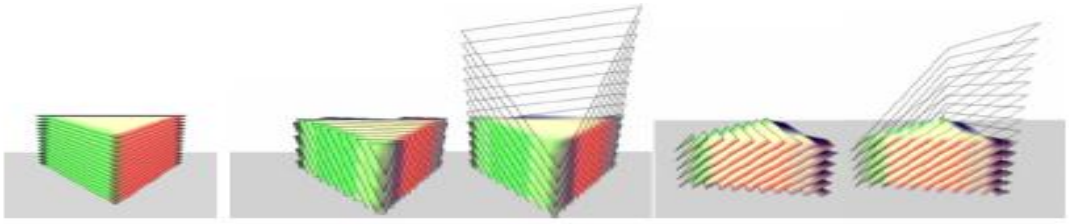


Figure 7 - Left: a regular texcell. Middle: a silhouette texcell (facing the camera) with and without clipping of the empty top. Right: a side view of it. [5]

3.3. RENDERING OF FOREST SCENES [6]

Paul Guerrero describes two methods for rendering a high scale forest and conducting a comparison. The methods are the Imposter-Method [6] and the Static Geometry Method [6]. In rendering forests scenes there are several problems that are encountered. First, rendering forests with a high level of detail without simplification will be very slow. Second, each tree requires rendering of thousands of polygons per frame [6]. The previously stated methods resolve these issues.

The main concept for the Imposter-Method is to store the scene in an octree, and for each octant in the octree an imposter has been built. An imposter contains six colors [6], and each color represents each one of the six sides of the octant. In order to render the scene, the octree is scanned and each octant that is small enough or is a leaf, is displayed on the screen. In this way, octants are displayed instead of the geometry. Furthermore, the color of each octant is based on the average of the colors of the six octants "sons". Or, in the case of octant leaf, the color is based on the average of the colors of each of the six sides.

Static Geometry Method is a method which renders billboard trees at far distances. If each tree is rendered separately, it will cause a large number of render calls, which it is not efficient. The paper suggests an approach which renders the relevant billboard trees according to the scene. The main concept is to use an octree data structure, where the billboard trees are stored. According to the distance to the camera, the relevant level of the octant will be displayed.

Finally, there are some problems with shading when using cross billboards in order to render trees. For example, billboards lack depth compared to fully geometrical trees, and as a result the shadowing does not emphasize the depth of the rendered trees. The paper suggests calculating the tree shading according to the tree depth and light source direction, in order to get a sense of depth to the billboards.

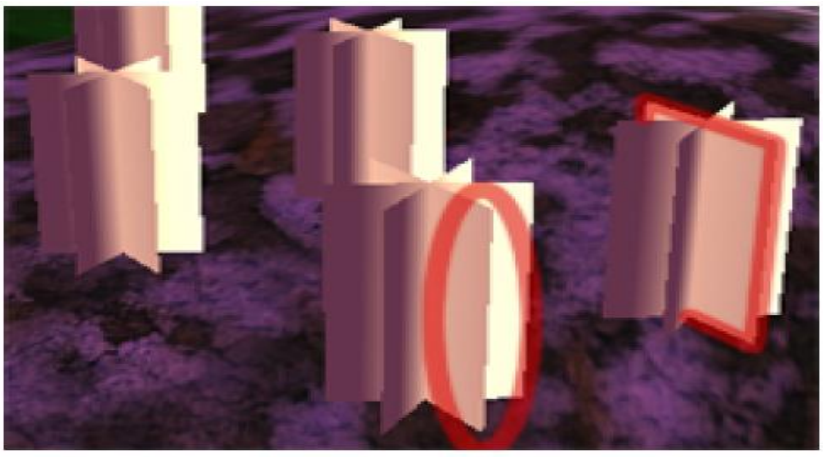


Figure 8 – An example of cross billboards. [6]

3.4. MULTIRESOLUTION FOLIAGE FOR FOREST RENDERING [7]

Real–Time rendering of different plants contribute to the realistic nature of computer graphic generated scenes. The main challenge of rendering plants in real-time is the amount of data, which needs to be displayed because of the 3D geometric structure they possess. The paper presents an algorithm for rendering a forest in a fast way. The algorithm has three main characteristics in that it is fast, it displays different levels of details of the plants and it contains vast details of the 3D geometric structure of the plants.

The algorithm handles the rendering of different leaves in the scene. The researchers develop two algorithms, Broad Foliage Simplification Algorithm (BFSA) [7], and Thin Foliage Simplification Algorithm (TFSA) [7]. The BFSA is for rendering broad leaves [7] which have different shapes and TFSA is for rendering thin leaves. Furthermore, the researchers elaborate on how these algorithms work with the LOD model. Finally, the paper presents a model for rendering forests. This model combines different elements in order to create realistic forests. The elements are: the algorithms that were previously mentioned, branch simplification method [7], plant distribution [7] and plant modeling method [7].

3.5. VEGETATION FOR TERRAIN GRAOHICAL MODELING USING L-SYSTEMS OVERVIEW [8]

L-System is the grammar for building a model, which defines the growth process of different plants and trees. The main concept for constructing plants and trees is to define a set of assumptions on the initial state and set of recursive rules, which define the growth of the plants and trees.

L-System: Simple Example for Node Rewriting
Axiom : FX
Rule : X= +F-F-F+FX
Angle : 45

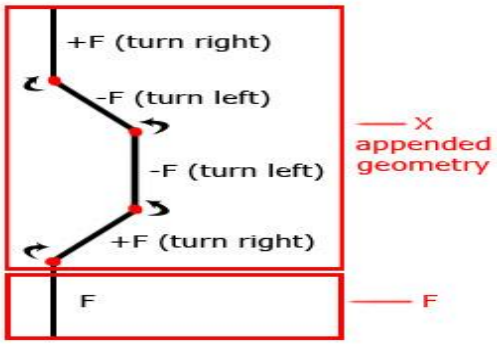


Figure 9 - A Simple L-System Example - Node Rewriting [8]

One of the main features of the modeling trees is to define the construction of the branches. By using an L-System, we can define the grammar in order to characterize the branches for modeling a tree. For example, this would include a set of rules defining the sizes of the branches, the maximum number of branches etc. More developed L-Systems include rules, which describe the impact of the environment on the plants and trees, for example, the season and amount of water.

4. DESIGN AND IMPLEMENTATION

4.1. Generation of Forests on Terrain

In order to generate forests, there are different factors that need to be defined. These include: the scale of the generated forests, the type of the trees, the 3D geometric structure of the trees and the amount of details to be rendered for each object according to the distance from the camera position. This section presents a method for generating forests on different terrains.

"Forest_Cell dots[N][N]" is an array data structure which represents the terrain and forest that have been rendered, on the size of N×N. The following example of code is the "Forest_Cell" data structure:

```
struct Forest_Cell {

    float x, y, z, type, shadow; };
```

The variables "x, y and z" are coordinates of a point in the 3D world. The variables "type" and "shadow" represent the type of tree to be rendered and the amount of exposure to light source, respectively. Figure 10 is an image that displays rendered terrain-net base on the "Forest_Cell dots[N][N]" array data structure. Each quad is constructed from the coordinates of four dots. For example the quad in the right button is constructed from the coordinates of dots[0][0], dots[0][1], dots[1][0], dots[1],[1]. In each quad, different trees will be rendered. The relevant data regarding the type of the tree and the amount exposure to the light source will be stored in the dot[0][0] which is located in the bottom right of the quad.

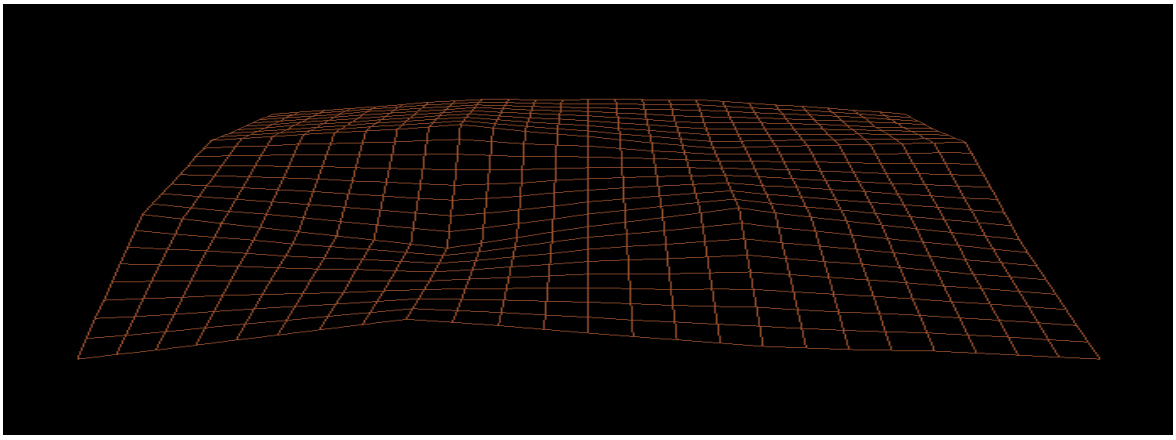


Figure 10 – Terrain-net image based on the "Forest_Cell dots[N][N]" array data structure.

This paper uses the regular texcells which have been created in the research paper Rendering Forest Scenes in Real - Time [5]. The regular texcells are based on the 2D textures that are created by a method in which a 3D volume dataset is sliced in a parallel fashion. There are 32 different textures that can be generated from the 3D dataset.

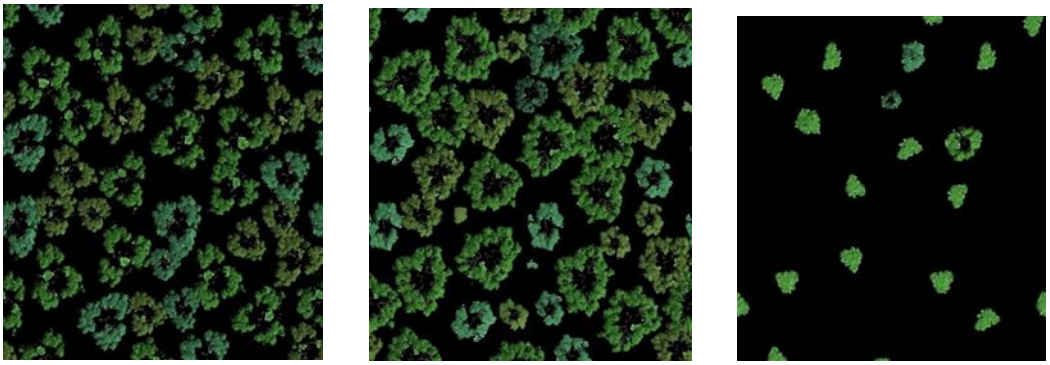


Figure 1 - From left to right: nuts_aper_0007.tga, nuts_aper_0016.tga, nuts_aper_0028.tga

The regular texcells are from the file format "tga". This type of file format ends with ".tga" [9]. It stores image data with 8, 16, 24, or 32 bits of precision per pixel. These images are constructed from a maximum of 24 bits of RGB and an 8-bit alpha channel [9].

Several manipulations need to be made on the regular texcells [5], in order to render specific trees instead a set of trees as shown in the figure above. From each regular texcell, only the parts that contain the specific tree will be rendered. As mentioned above, there are 32 textures that are generated from the 3D volume dataset. As the number of the image decreases, it represents a lower layer of the 3D dataset. As a result, the textures in the middle contain the central part of the tree. This tends to be the biggest part of the tree. By defining the boundaries of the relevant parts that need to be rendered based on the middle textures, the same boundaries will be relevant to the other textures. The method for rendering the relevant parts is to define the coordinates of a quad which contain the image of the tree. The following example of code is for defining texture coordinates:

```
glTexCoord2f (0.25f, 0.05f); // top left vertex

glTexCoord2f (0.40f, 0.05f); // bottom left vertex

glTexCoord2f (0.40f, 0.2f); // bottom right vertex

glTexCoord2f (0.25f, 0.2f); // top right vertex
```

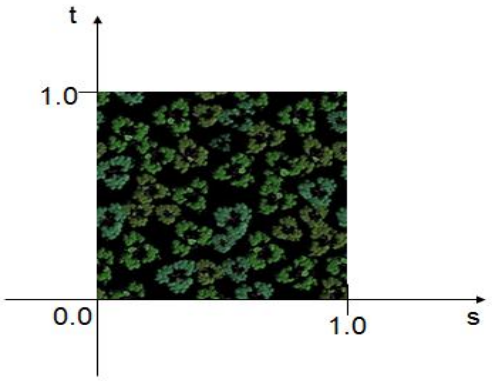


Figure 11 – An example of Texture coordinates.

There are 12 different textures that are needed in order to render 3D trees. They are as follows: "0006.tga", "0008.tga", "0010.tga", "0012.tga", "0014.tga", "0016.tga", "0020.tga", "0024.tga", "0026.tga", "0027.tga", "0028.tga", "and 0029.tga". There are three reasons for using only 12 textures:

- Rendering fewer textures makes the rendering process more efficient.

- As the number of the image decreases, it represents a lower layer of the 3D dataset. By choosing specific textures from the 32 options, a 3D tree can be rendered with fewer textures. When using this method, it important to progress in a decreasing order of the textures in order to render the tree correctly.

- Instead of calculating different sets of textures according to the distance of the camera position, there is only one set of textures assigned to all distances from the camera. By rendering the textures mentioned above, the quality of the 3D tree image is preserved from different distances from the camera position. This method prevents the need to calculate the distance for each tree which increases the efficiency of rendering the scene.

The method for creating a 3D tree on the terrain is constructed from three main steps. The first step is to define the type of tree to be rendered. Each cell in the array data structure has a variable "type". The variable is an integer number generated by a random function, while each number symbolizes a different type of tree. The following example of code is for generating a random integer number in the range of 0 to 4:

```
dots[w][z].r = rand()%5;
```

In order to render specific trees to different parts of the terrain, such as valleys, planes and mountains, different random functions are defined.

The second step is to map the different textures to coordinates on the terrain. Each tree in a texture has specific coordinates. These coordinates are mapped to coordinates of different quads that construct the terrain. The following code example is for mapping texture to a quad:

```
glColor4f(1.0,1.0,1.0,1.0);
glBegin(GL_QUADS);
//glNormal3f(v.x,v.y,v.z);
// top left vertex
glTexCoord2f(0.25f, 0.05f);
glVertex3f(dots[k1][z1].x, dots[k1][z1].y+h1, dots[k1][z1].z);
// bottom left vertex
glTexCoord2f(0.40f, 0.05f);
glVertex3f(dots[k1+1][z1].x, dots[k1+1][z1].y+h1, dots[k1+1][z1].z);
// bottom right vertex
glTexCoord2f(0.40f, 0.20f);
glVertex3f(dots[k1+1][z1+1].x, dots[k1+1][z1+1].y+h1, dots[k1+1][z1+1].z);
// top right vertex
glTexCoord2f(0.25f, 0.20f);
glVertex3f(dots[k1][z1+1].x, dots[k1][z1+1].y+h1, dots[k1][z1+1].z);
glEnd();
```

The method for rendering a 3D tree is to map each one of the tree textures from the set of 12 textures to the same “x” and “z” coordinates but different “y” coordinates. The lowest layer texture "0006.tga" is mapped to the “y” coordinate in the array data structure. As the number of the layer texture increases, the “y” coordinate increases. For example: "0006.tga" is mapped to the “y” coordinate, "0008.tga" is mapped to the “y+1” coordinate, "0010.tga" is mapped to the “y+2” coordinate, and so forth.

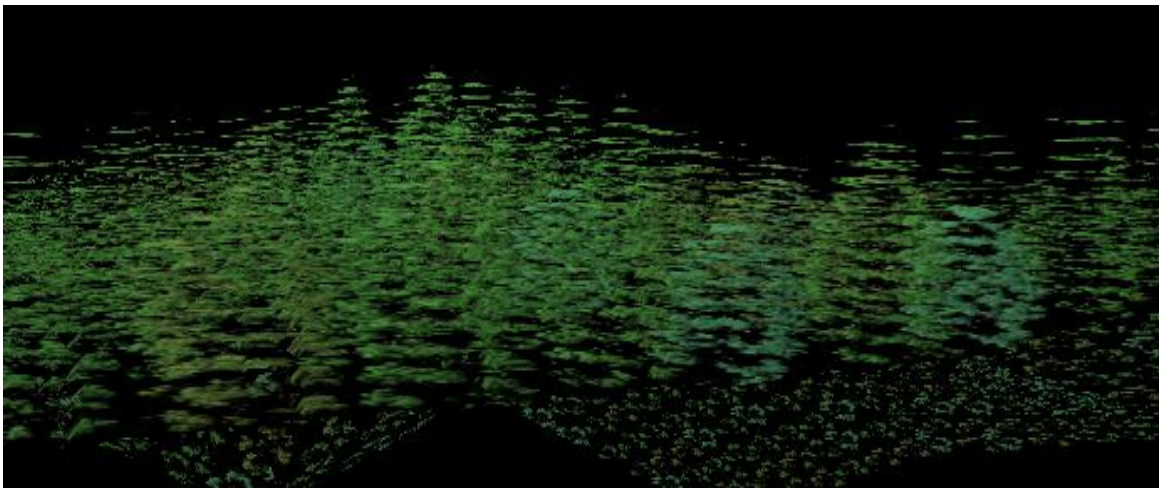


Figure 12 – This image exhibits the layers of textures of the 3D trees.

Based on the length and width of each quad in the terrain, the distance between the layers of textures is defined. In order to get more realistic 3D rendering, each type of tree needs to fulfill the following condition:

(Length of a quad) / (Width of a quad) ≈ 1

(Height of a tree) / (Width of a quad) ≈ 1

The length and width of each quad is defined to be 10 in the 3D world coordinates. The height of a tree in each quad is 11. In the research corresponding to this paper, they are the approximately the same size. In order to add different types of trees to the forest, the above condition may be changed for each type of tree. The condition is changed according to the size of the texture, which contains the image of the tree. This will create more realistic 3D trees.

The third step is to manipulate the transparency and color of each of the 12 textures which construct the 3D tree. The color variables control the brightness of the textures, which is useful to implement shadow. The Alpha variable controls the transparency of the texture. By controlling this variable, different textures can be blended together, which creates different effects in computer graphics. After several tests, in which the Alpha variable has been changed for each texture, the value 1.0 for each texture creates the best image for a 3D tree. If the textures have different Alpha values, some part of the 3D tree will be more transparent. As a result, it could negatively impact the realistic appearance of the tree.

4.2. Dynamic Lighting and Shadowing

Lighting and Shadowing are features in computer graphics that contribute to the realistic feeling of a 3D scene. Furthermore, in a 3D scene it contributes to the visual depth of the scene and the visual shape of different objects in the scene. The lighting and shadowing are created by calculating the light source position and its type. The lighting and the type of material of different objects in the scene influence the lighting and shading that are viewed from the camera position. This paper proposes a method for rendering lighting and shadowing based on several manipulations of the textures, which construct the different trees in the terrain. The first part describes the method for rendering the light on each tree according to the light source position that mimics the sun. The second part describes the method for creating shadows according to the varying elevation of the terrain while taking into calculation the height of the trees.

The first part for rendering the light on each tree according to the light source position is based on several assumptions:

- Because the light source mimics the sun, the rays emanate from the light source position, they do not lose energy as they travel longer distances and they are parallel. Furthermore, the light source mimics the path of the sun in the real world, which influences the realistic feeling of the scene.
- The calculation of the light does not take into consideration the Diffuse Reflection and Specular Reflection.
- The calculation is based on each one the quads, containing only one tree.

The main concept for rendering the light is that the parts of the tree that are facing the sun will be brighter than other parts of the tree, such as the back of the tree.

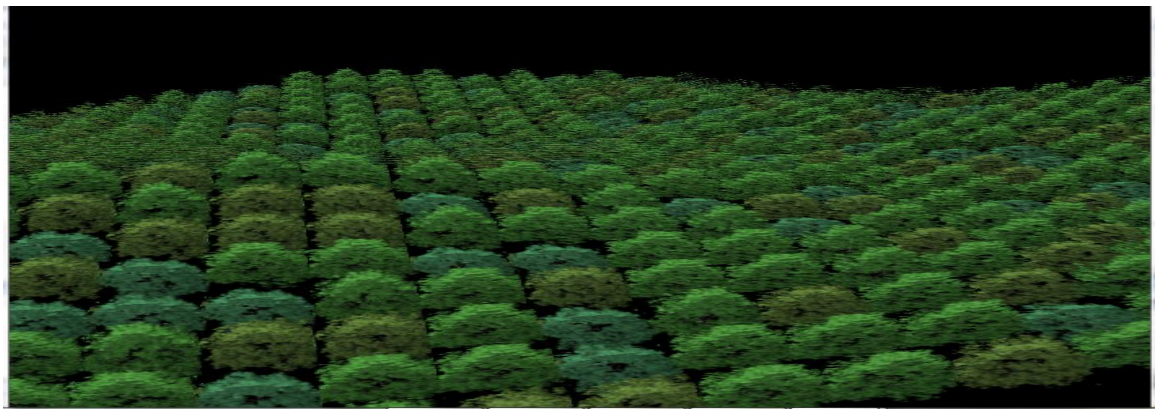


Figure 2 – The location of the light source is angled towards the position of the camera.

In the figure above, the light source is towards the position of the camera. As a result, the parts which are not facing the sun are darker than the facing part, which can be noticed in the top part of the trees in the image above.

The method for creating this effect is to slice the textures into two parts and change their color according to the sun's location. The slicing method needs to be the same for all 12 different textures which construct the tree. By implementing it to all 12 textures and modifying the colors accordingly, one part of the 3D tree becomes brighter while the other part becomes darker. In order to split a texture, two sets of coordinates needed to be defined. By rendering each part with different colors, the light affect has been achieved. The following code example is for splitting textures into two parts:

A whole texture

```
// top left vertex
glTexCoord2f(0.25f, 0.05f);
// bottom left vertex
glTexCoord2f(0.40f, 0.05f);
// bottom right vertex
glTexCoord2f(0.40f, 0.20f);
// top right vertex
glTexCoord2f(0.25f, 0.20f);
```

First part of the texture

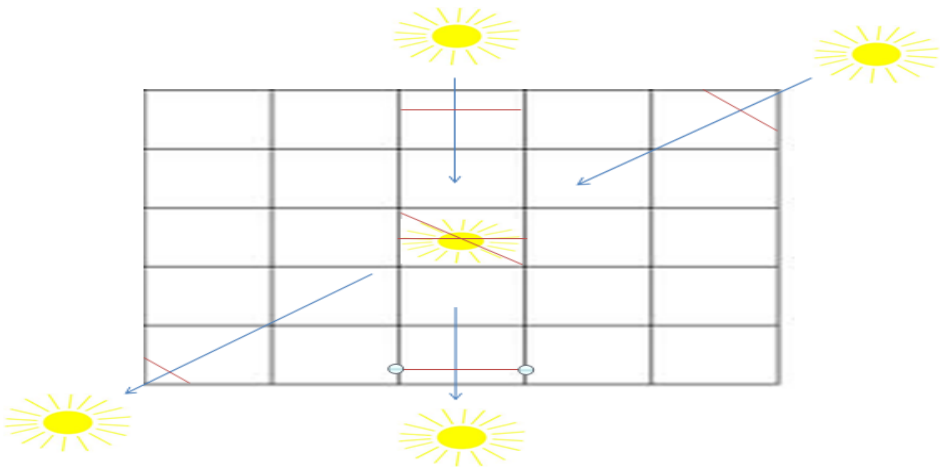
```
// top left vertex
glTexCoord2f(0.25f, 0.05f);
// bottom left vertex
glTexCoord2f(0.40f, 0.05f);
// bottom right vertex
glTexCoord2f(0.40f, 0.125f);
// top right vertex
glTexCoord2f(0.25f, 0.125f);
```

Second part of the texture

```
// top left vertex
glTexCoord2f(0.25f, 0.125f);
// bottom left vertex
glTexCoord2f(0.40f, 0.125f);
// bottom right vertex
glTexCoord2f(0.40f, 0.20f);
// top right vertex
glTexCoord2f(0.25f, 0.20f);
```

After splitting the texture into two parts, the color for each one needs to be defined. In the research corresponding to this paper, the color for the brighter side is defined as glColor4f(1.0,1.0,1.0,1.0), while the color for the darker side is defined as glColor4f(0.6,0.6,0.6,1.0). Furthermore, during the process of the research, the approach for modifying the transparency variable has been tested, but the results were not of good quality for creating the light effect.

In order to split the texture which contains the tree image, according to the position of the sun, the relationship between them needs to be defined. There are two routes that the sun can follow. The first path is when the sun shines vertically and the slicing method is horizontal. The second path is when the sun shines diagonally and the slicing method is also angled. The following image exhibits the paths of the sun.



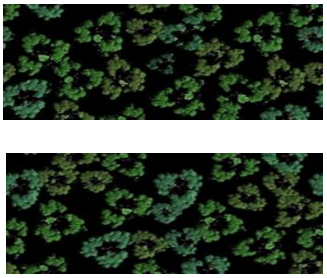


Figure 13 – This image exhibits the paths of the sun and a sliced texture.

The relationship between the location of the sun and the ratio of the slicing is based on several assumptions for both routes:

- The starting and ending position of the sun mimics the time before sunrise and after sunset respectively.
- The middle position of the sun mimics the time when the sun is directly above the forest.
- The amount of sun locations between sunrise to the middle of the day influence the number of slices. For example:
  0°->a= all texture darker, 22.5°->1/4 texture with light, 45°->2/4 texture with light, 67.5°->3/4 texture with light, 90°->all texture light
- The texture mapping for each part of the texture according to the sun position needs to be mapped to the right coordinates onto the terrain. These coordinates need to have the same slicing ratio in order to preserve the 3D tree image. The following code example is for mapping half of the texture to one half of a quad on the terrain. As is visible from the following code, there are manipulations of the coordinates “y” and “z” in order to create the correct mapping.

```
glBegin(GL_QUADS);
//glNormal3f(v.x,v.y,v.z);
// top left vertex
glTexCoord2f(0.25f, 0.05f);
glVertex3f(dots[k1][z1].x, dots[k1][z1].y+h1, dots[k1][z1].z);
// bottom left vertex
glTexCoord2f(0.40f, 0.05f);
glVertex3f(dots[k1+1][z1].x, dots[k1+1][z1].y+h1, dots[k1+1][z1].z);
```

```
// bottom right vertex
glTexCoord2f(0.40f, 0.125f);
glVertex3f(dots[k1+1][z1+1].x,((dots[k1+1][z1].y+dots[k1+1][z1+1].y)/2)+h1,
dots[k1+1][z1+1].z+5);
// top right vertex
glTexCoord2f(0.25f, 0.125f);
glVertex3f(dots[k1][z1+1].x,((dots[k1][z1].y+dots[k1][z1+1].y)/2)+h1,
dots[k1][z1+1].z+5);
glEnd();
```

The algorithm of the second part for creating shadows according to the varying elevation of the terrain is constructed from two concepts. The first concept is the path of the sun, from sunrise to sunset. As mentioned before as the rays emanate from the light source position, they do not lose energy as they travel longer distances and they are parallel. Based on the assumptions, the algorithm assumes that there is a ray which passes through each column or diagonal slice according to the path of the sun. The following figure exhibits the path of the sun and the direction of the rays. The black arrows describe the sun's paths. The red and blue arrows describe the ray's direction based on the path and location of the sun.

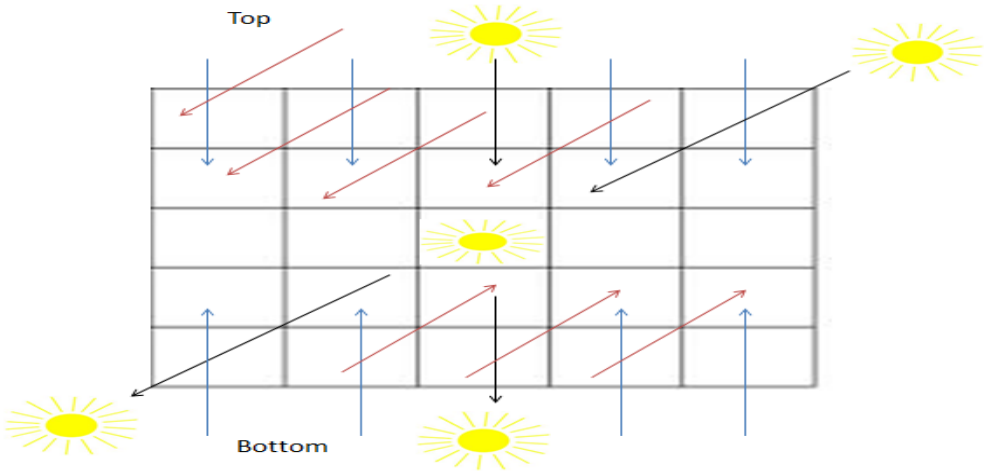


Figure 14 – This image exhibits the paths of the sun and the direction of the rays.

The method for scanning the "Forest_Cell dots[N][N]" array data structure is defined according to the rays directions. This allows for rendering of two main features:

- The parts of the tree that face the sun.
- The amount of exposure of each tree according to the varying elevation of the terrain and the ray's direction.

The following code example is for scanning the "Forest_Cell dots[N][N]" array data structure from Top to Bottom and the opposite direction.

```
Top to Bottom: for (int k=0; k<25; k++){
                    for (int z=1; z<25; z++){
                        ……..
                        for (int t=0; t<z; t++){
                            ……. }
                    }
                }

Bottom to Top: for (int k=0; k<25; k++){
                    for (int z=23; z>-1; z--){
                        …….
                        for (int t=24; t>z; t--){
                            ……..}
                    }
                }
```

The second concept for creating shadows according to the varying elevation of the terrain is to calculate the amount of exposure to light of each tree. This can be done in several steps:

1. Input: sun location (sun angle), the tree height according to the terrain.
2. Calculate the linear equation based on the input.

3. For each tree that lies in the path between the sun and the input tree (the same column of the input tree), execute the following steps:
    a. If both the coordinates of the top and bottom of the tree are entered into the equation and result in positive values, define the variable "shadow" of the input tree as 0. This indicates that the input tree is totally blocked from the sun.
    b. If the coordinate of the top of the tree is entered into the equation and results in a positive value, while the coordinate of the bottom of the tree results in a negative value, define the variable "shadow" of the input tree as 0. This indicates that the input tree is totally blocked from the sun.
    c. If both the coordinates of the top and bottom of the tree are entered into the equation and result in negative values, calculate the perpendicular distance between the coordinates of the top of the tree to the equation. Then define the variable "shadow" of the input tree to the perpendicular distance.
4. If the value of the "shadow" variable of the input tree is equal to zero, change all brighter parts (parts which are exposed to the sun) of the tree to a darker color. If the value is not zero execute the following steps:
    a. If the value is larger or equal to 11, do not modify the brighter parts of the tree. This indicates that none of the trees in the path blocks the input tree.
    b. If it is less than 11, modify the (11-"shadow") layers from the bottom of the tree to a darker color. The "shadow" variable indicates how many layers from the top of the input trees are expose to the sun.

The rendering of the amount of light that each tree is exposed to, defines how many layers from the bottom of the tree will not be exposed to light.

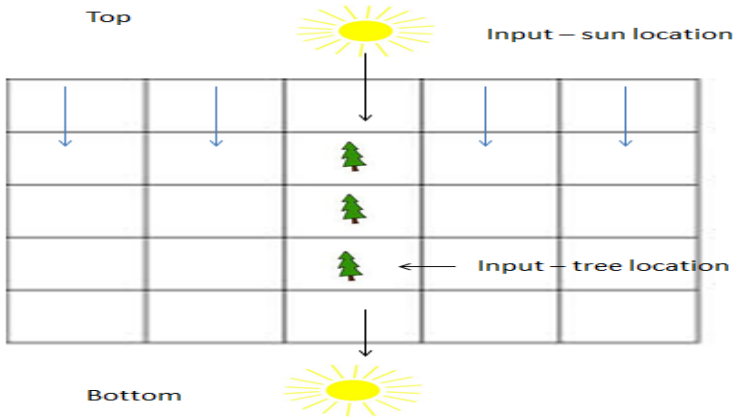


Figure 15 – This image exhibits the trees that lie along the path between the input tree and the sun.

4.3. Code Structure for rendering efficiency

In order to improve the efficiency of rendering the scene, several improvements can be implemented. The implementation includes the simplicity of the code, simplicity of different calculations and using the features of OpenGL.

Some improvements are implemented with the textures which construct the trees. Each tree is constructed from 12 textures which render different layers of the tree. By choosing the right textures, 3D trees can be rendered with less than 32 textures, which are created from the 3D volume data set. As a result, the algorithm saves 24 uploads of texture into the texture memory by not using the 24 textures. Another advantage for using 12 textures is that a 3D tree can be rendered with good quality from different distances from the camera position. This concept prevents calculation for deciding how many layers to render according to the distance for each tree.

Moreover, by operating all the relevant functions related to the different objects which use the specific texture, the rendering efficiency is improved. This method is the more efficient one because each texture has been uploaded once instead of each texture being uploaded several times. For example: if there are 25 object of trees, there will be 12 called for uploading the texture instead of 25x12. Furthermore, by limiting the slicing method for two parts in order to implement the lighting and shadowing, a set of texture mapping has been prevented. The less efficient method is to slice the texture into several pieces in order to create more realistic lighting and shadowing, but it will increase the texture mapping functions.

5. CONCLUSION AND FUTURE WORK

I have described a method for rendering lighting and shadowing in real-time on 3D trees which have been constructed from textures. Furthermore, I also described how several modifications of the code can create more efficient rendering.

This method allows us to implement high scale forest scenes with dynamic lighting and shadowing. The paper calculates in real-time the light and shadow for each tree which contributes to the realistic feeling of a 3D scene. The shadow effect can be seen from different angles and ranges of the camera, for example: high scale forest point of view or a group of trees from close range with different angles. Furthermore, from several camera positions the trees which are close to the camera lose the 3D effects, as a result of using regular texcells. They are created by a method in which a 3D volume dataset is sliced in a parallel fashion, which impacts the 3D effects of trees in close range.

The approach of this method is different from prior research papers in several ways. First, there is a set of 12 textures that are used for the various distances of the camera position. Second, the light and shadow is implemented by changing the color of a set of textures which contributes to the feeling of 3D effects. The papers "How to Render a Forest in 3D" [4] and "Rendering of Forest Scenes" [6] render textures with images of trees. According to the distance of the camera position, each texture is rendered with a varying amount of detail. The significance of this paper is that each tree is constructed from a set of 12 textures. This set of textures is rendered for all the different distances from the camera positions. Furthermore, implementing light and shadow on texture is a challenging subject in computer graphics, especially with complicated geometric structure of trees. The paper "Rendering of Forest Scenes" calculates the shadow according to the light source and the depth, while in my method the light and shadow are calculated as a set of textures which impact on the rendered 3D tree.

Some of the different approaches in the prior research papers can be combined with the current approach in order to improve the realistic nature of the forest. An example of this would be defining a L-system grammar for the location of trees on the terrain. This grammar can include the elevation of the terrain and the weather which impacts the location and type of trees.

For future work, there are several improvements that could be implemented:

- To add different types of trees, such as: trees with various heights or structures, in order to create different types of forests. Furthermore, a specific slicing method is needed to adapt to the newly added tree model. This will contribute to the realistic light and shadow rendering.
- Rendering lighting and shadowing according to any light source position and not only according to specific paths of the sun.
- The rendered trees in the forests are structured in rows and columns. As mentioned in the paper, the structure is related to the method of calculating the light and shadow. A method which places trees on the terrain in a random manner, and calculates the lighting and shadowing according to the location, will improve the realistic feeling of the forest.

6. REFERENCES

[1] "cplusplus.com", Retrieved November 20, 2011, from www.cplusplus.com.

[2] Song Ho Ahn. "OpenGL", Retrieved November 20, 2011, from www.songho.ca/opengl/index.html

[3] Samuel R.Buss. "3-D Computer Graphics", New York: Cambridge University Press, p 67 – 87, 2003.

[4] Hans Häggström. "How to Render a Forest in 3D", March 2001.

[5] Ph. Decaudin, F. Neyret. "Rendering Forest Scenes in Real-Time". Rendering Techniques '04 (Eurographics Symposium on Rendering), p 93 - 102, June 2004.

[6] Paul Guerrero. "Rendering of Forest Scenes", Technical University of Vienna, September 2006.

[7] Qingqiong Deng, Xiaopeng Zhang, Gang Yang and Marc Jaeger. "Multiresolution foliage for forest rendering", Computer Animation and Virtual Words 2010, May 2009.

[8] Przemyslaw Prusinkiewicz, Aristid Lindenmayer. "The Algorithmic Beauty of Plants", New York: Springer-Verlag, p 1 -70, 2004.

[9] Truevision TGA. "FILE FORMAT SPECIFICATION Version 2.0", Truevision, Inc., p 1 – 3, January 1991.