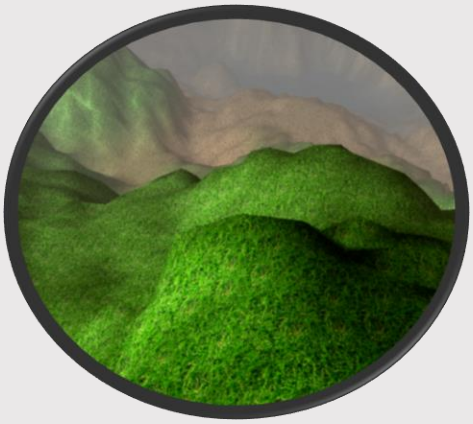
NATURAL SCENE SIMULATION



BRIEF DESCRIPTION

The simulation of natural scenes is a very important branch of Computer Graphics and has broad application fields. The main idea is to simulate a real-world natural scene, by drawing island, trees, grass and simulate the daylight, water movement and surround the world with a realistic skybox and some special effects like: fog, snow falling, … etc.

TABLE OF CONTENT

MINIMUM REQUIREMENTS

[SkyBox]
Draw a skybox with a realistic 6 textures one for each face of the cube.

[Terrain]
Given any height map draw a multi-texture terrain to simulate the island.

[Light] Simulate both the day and night light.

[Trees and Grass]
Draw 3D tree models in random locations of the terrain and draw billboards with grass texture in different locations.

[Water] Simulate the sea motion.

[Special effect] Apply Fog effect.

[Camera]
Implement First Person Camera that can walk on the surface of the terrain accurately.

[Tour]
By pressing a certain key, allow the camera to take a tour around the scene automatically.

BONUS

[Sound effect] add sound the simulate the sound of beach & water.

[Special effects] add much complicated effects like snow, rain, … etc.

[Water effect] add reflection & refraction to water.

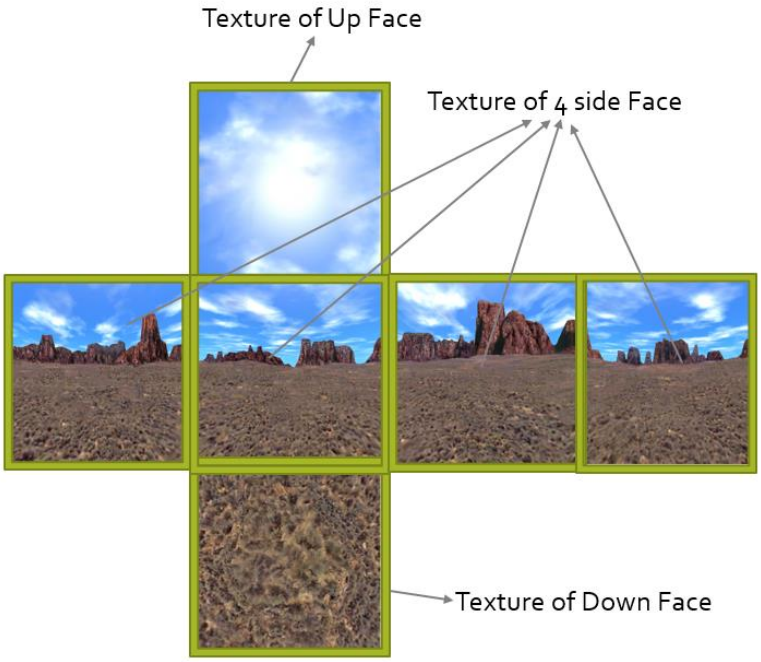
[Save & Load Scene] add option to save an entire scene and loading it.

[Human] add animated 3D human model and implement 3rd person camera to follow the human in the scene.

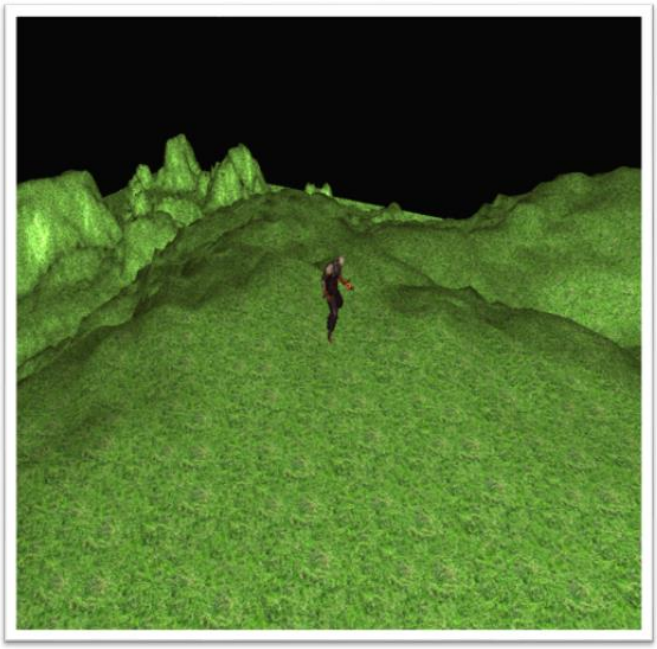
Detailed Description:

SKYBOX

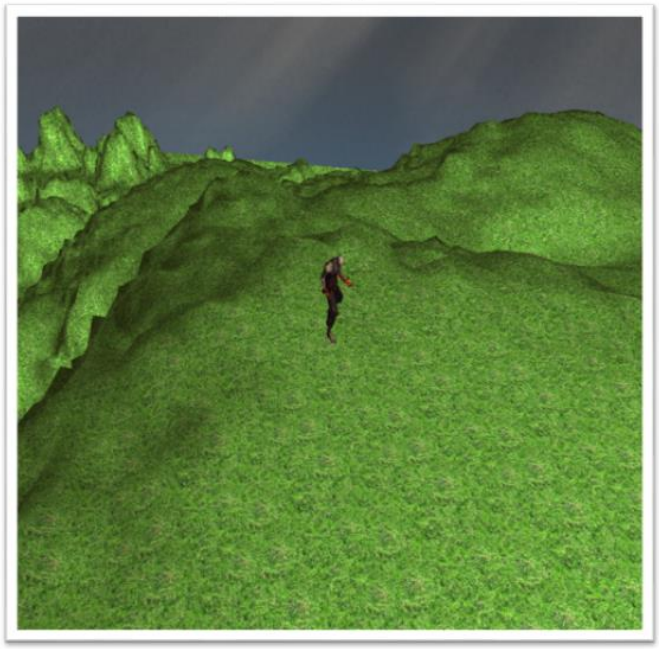
- To make the world appears more realistic, we need to draw a “SkyBox”.
- Sky box is a very big cube that surround the whole world, with a different texture on each face, for example:
  - Up face has the texture of sky.
  - Down face has texture of ground.
  - The other 4 faces have a side texture.
- When you look at it from inside, it looks so realistic better than that black space.



Without Skybox



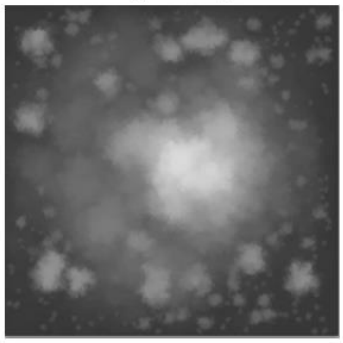
With Skybox



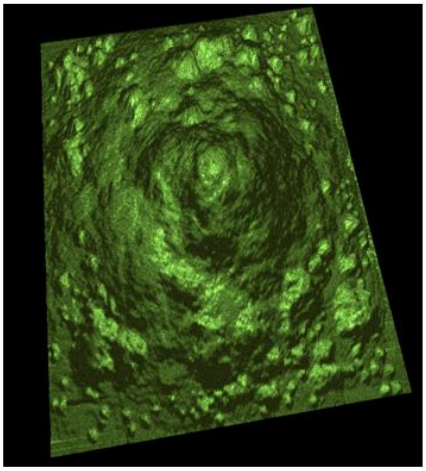
TERRAIN

- To draw a more realistic island, use Terrain.
- What is Terrain?
  - Using the same previous concept but build the world based on image not text file.
  - This image called “Height Map” a gray scale image where the gray value of each pixel defines the height in the world.
  - Imagine as if you are looking on island from above.
  - Dark gray means low heights (ground), bright gray means high heights (mountains).

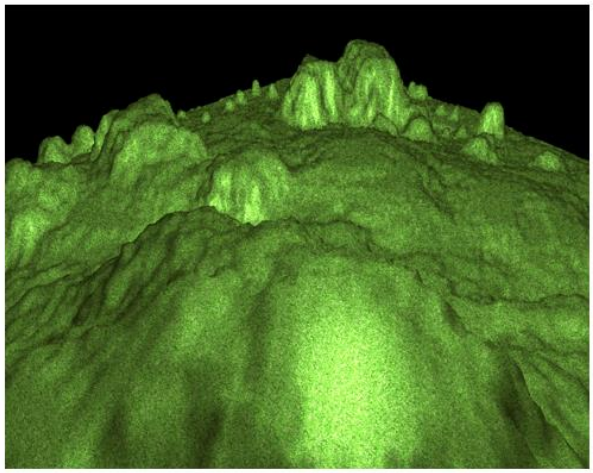
Height Map



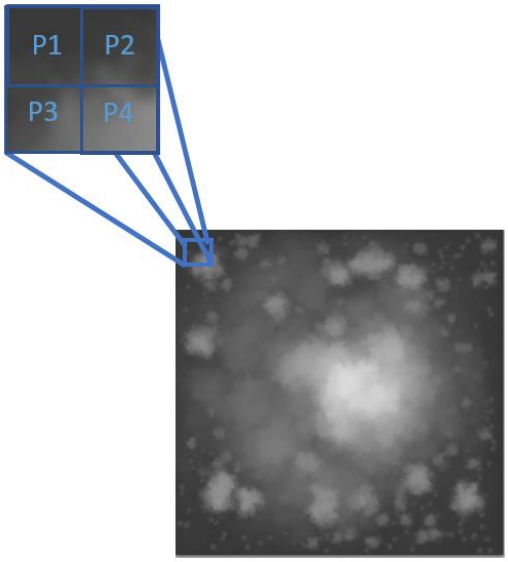
World from above



View from the highest area

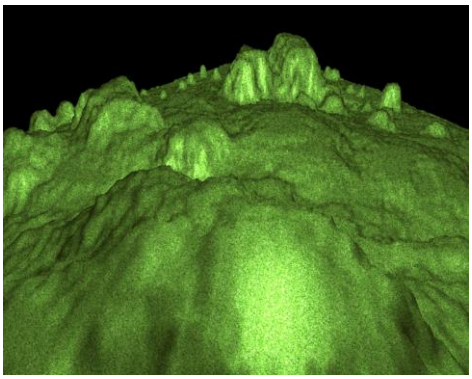


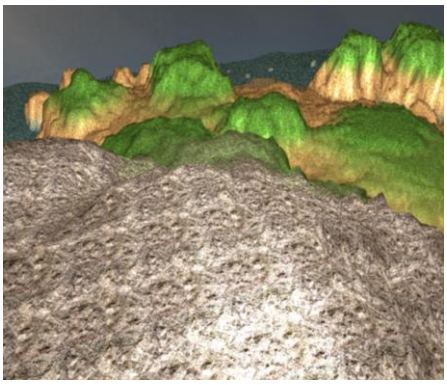
- How to draw a terrain “given a grayscale image (Height Map)”?
  - Open the image using (Bitmap) class in C# in namespace “System.Drawing”.
  - Any image is a 2D array of pixels, simply loop on the image pixel by pixel.
  - For each pixel, take a window of size 2x2 (i.e. at pixel with index (i,j), take (i+1,j), (i,j+1) & (i+1,j+1) the 3 neighbors of pixel (i,j)).



  - Given the 4 pixels create 4 vertices, vertex corresponding for each pixel.
  - Let i be the X value of vertex, j the Z value and finally the color of pixel is the Y value.
  - These 4 vertices are then used to draw 2 triangles.
  - Compute the normal for each triangle using the equation explained in light lab.
  - Add “U, V” data for each vertex.
  - Control the scale of the terrain in width, height & depth using scale matrix.

- **Multi-Texture Terrain**
  - Multi-Texture terrain is more realistic than single texture terrain





  - Let's say we have 4 textures (Sand, Grass, Rock, Ice).
  - Send all 4 textures to fragment shader and the position of the current vertex, the fragment shader should decide which texture to use based on the height of the vertex. Let's first normalize the value of Y of the vertex by dividing it by the maximum height, this will produce a value for Y ranged from 0~1 in each vertex.
  - If Y ranged from:
    - 0~0.25 then use Sand Texture.
    - 0.25~0.5 then use Grass Texture.
    - 0.5~0.75 then use Rock Texture.
    - 0.75~1 then use Ice Texture
  - To make the transition from texture to another smooth the function **mix (color1, color2, mixFactor).**

TREE

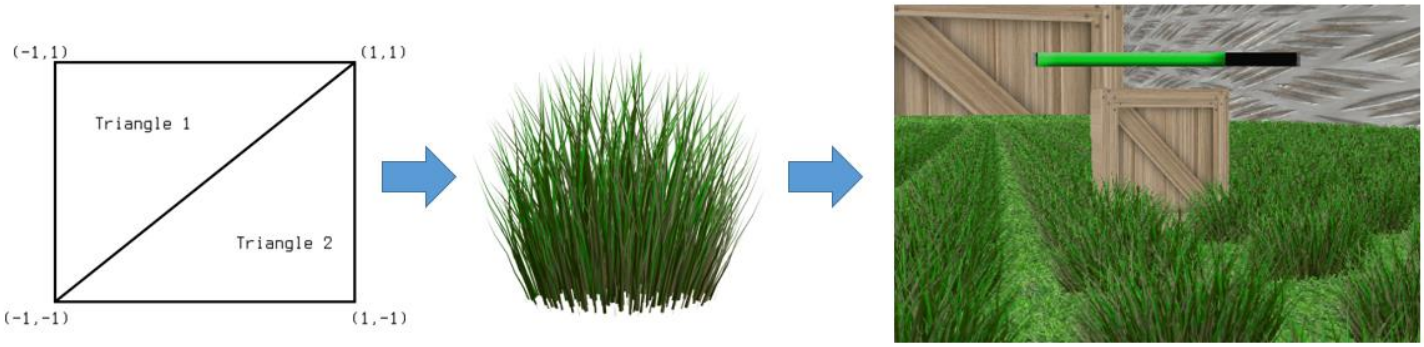
- Trees are a static 3D model.
- Draw it in random location of the terrain but take care of the height of this location.

GRASS

Grass can be rendered as a **billboard** instead of 3D model to reduce computations done by drawing 3D models.

A **billboard** is a flat object, usually a quad (square) in a 3D space, and can be done as follows:

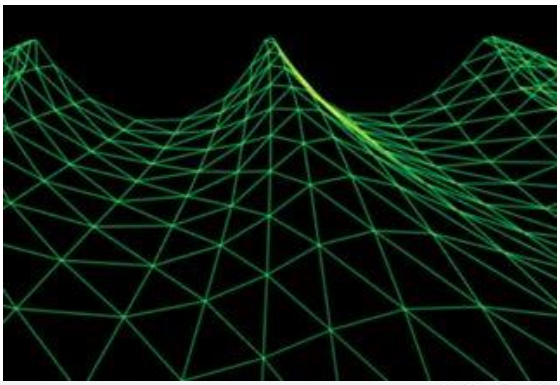
- First draw a quad (2 triangles).
- Load Texture and apply it on the quad.
- Then draw it in a certain position in the 3D world.



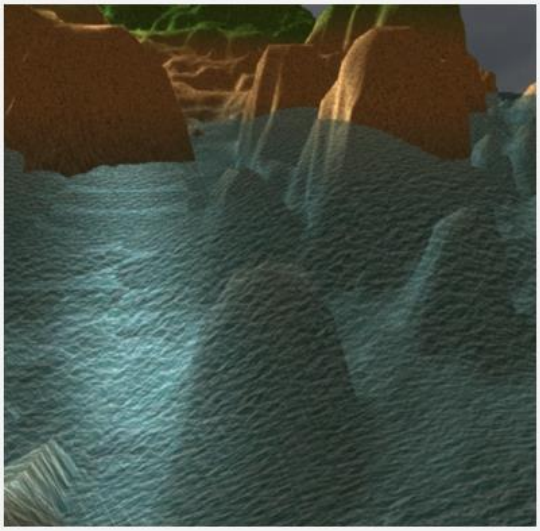
WATER

- Draw a 2D grid of squares (the same way we created the terrain consists of multiple squares, the only difference is that all squares in water initially have the same Y = 0).
- Add water texture, UV data & compute normal.
- In **vertexshader**, manipulate y (height) of each vertex using this equation:
  - Vertex.y += (sin (Time*Vertex.x) + cos (Time * Vertex.z)) * waveAmplitude;
  - Time is a float variable that increase with time in Renderer and then sent to VertexShader, wave amplitude is the height of water wave.
- In **fragmentshader**, manipulate the alpha color component
  - out vec4 color;
  - In the end of **fragmentshader**: color.a = 0.5; //to make the 2D grid half transparent
- Note: Blending must be enabled to allow the transparency
- **To enable Transparency:**

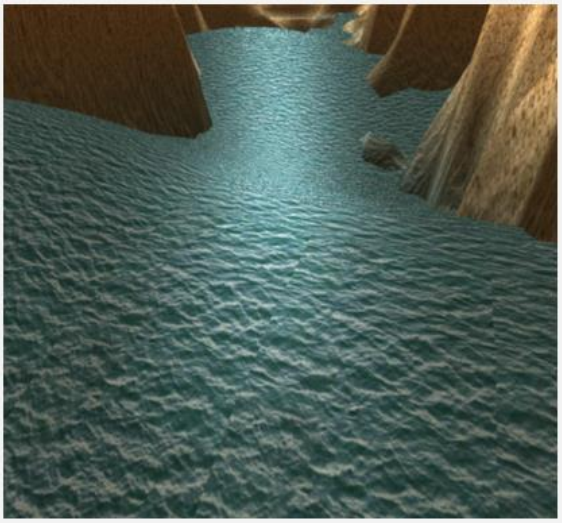
```
Gl.glEnable(Gl.GL_BLEND);
Gl.glBlendFunc(Gl.GL_SRC_ALPHA, Gl.GL_ONE_MINUS_SRC_ALPHA);
//Draw your Billboard then disable blend again
Gl.glDisable(Gl.GL_BLEND);
```



With alpha = 0.5



With alpha = 1 (no transparency)

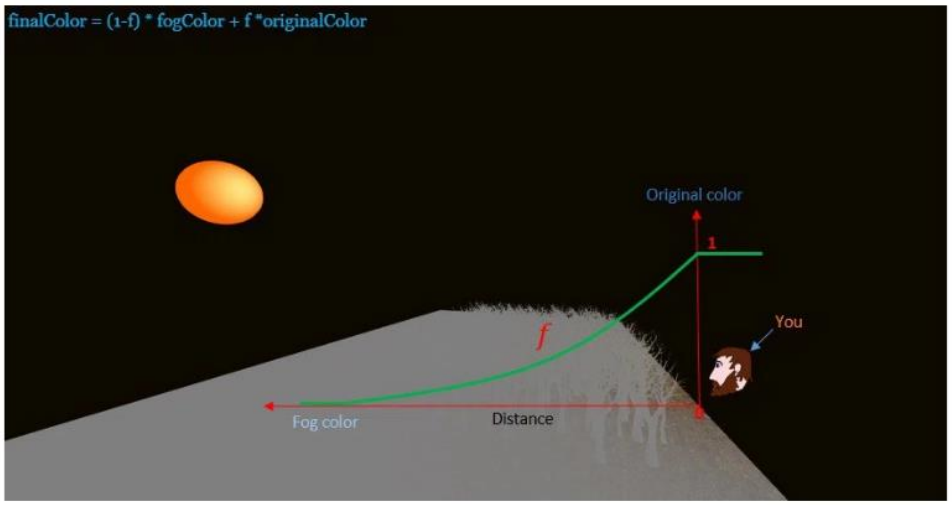


FOG

- Adds more realism to scene.
- Fog density increase as the distance increase.

$$f = e^{-d*b} = \frac{1}{e^{d*b}} \qquad \begin{array}{l} where\ d = distance, \\ b = attenuation\ factor\ or\ fog\ density \end{array}$$

Exponential Fog Formula



- To apply Fog, we need to send view matrix to vertex shader and pass viewspace = V*M*vertex to fragment shader (where V is view matrix and M is model matrix and vertex is the vertex position).
- In **fragmentshader**:
  - Define the fog color as vec3, fog density as a float value (ex: 0.05).
  - float dist = length(viewspace); //the distance of vertex from camera position
  - Apply fog equation such that attenuation is the fog density.
  - fogFactor = clamp(fogFactor, 0.0, 1.0);
  - color = mix(fogColor, color, fogFactor); //interpolate between color and fogcolor with fogfactor ratio.

TOUR

- Generate Random N points on the terrain.
- For each point, move the camera from point(i) to point(i+1).
- The motion should be smooth steps (i.e. don't make the camera disappear from point(i) and appear at point(i+1) instantly).
- Camera should move on its own (i.e. without any keypress) until the tour ends.