









Project #3













1 Requirements

Viewport This time, you do not need to implement two viewports. One viewport is enough.

Controlling the chooper The chopper still needs to be controlled by the keyboard.

- : Rotate counterclockwise
- : Rotate clockwise
- : Move forward
- : Move backward
-  or : Move upward
-  or : Move downward

Controlling We can change the view (camera) as follows. Refer to the demo video.

- + / +: Rotate the view around the z-axis of the world coordinates system.
- + / +: Tilt down/up the whole world.
-  or : Zoom in
-  or : Zoom out
- Note that the z-axis is always pointing upward in the screen.

The terrain Render the terrain using the image in Figure 1.

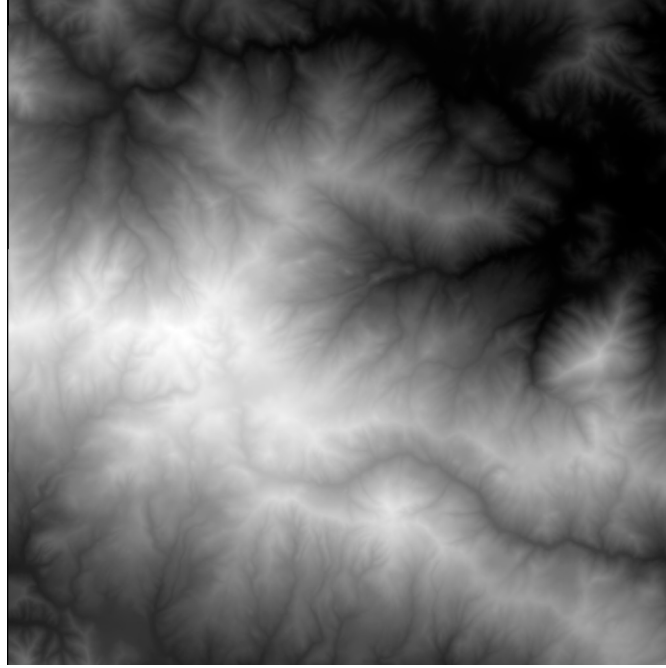


Figure 1

- On the host side, you need to generate a $N \times N$ quad grid with each quad is split into two triangles.
- Each vertex of the grid should have ONLY ONE attribute, the texture coordinates distributed evenly in $[0, 1] \times [0, 1]$. You SHOULD NOT use position/normal attributes. They should be computed in the vertex shader based on the texture. (Will be explained soon.).

This feature in fact is optional. The minimum number of textures accessible from the vertex shader, required by WebGL1 spec, is 0. (Read this) This can be checked by calling `gl.getParameter(gl.MAX_VERTEX_TEXTURE_IMAGE_UNITS)`. Or, you can go to WebGL Report and check the value of “Max Vertex Texture Image Units” in the “Vertex Shader” box.

- You also need to generate an index buffer for indexed rendering.
- When rendering the terrain, you need to bind the texture generated from the image in Figure 1.
- In the vertex shader, based on the texture coordinate attribute, you need to compute x , y , z coordinates as follows.
 - The x , y coordinates can be computed by simply scaling and translating (to center the terrain in the world) of the texture coordinates. For example, if we want the lengths of the terrain along the x - and y -axes be L_x and L_y , then

$$x(s, t) = L_x \cdot s - (L_x/2)$$

$$y(s, t) = L_y \cdot t - (L_y/2)$$

- The z coordinate needs to be obtained by fetching the texture. Scale it appropriately if needed.:

$$z(s, t) = S \cdot \text{texture}(s, t)$$

- The tricky part is to compute the normal vector. (reference: Unit normal vector of a surface)
 1. The normal vector at the point p is orthogonal to the tangent plane at p .
 2. For a smooth parametric surface $f(s, t) = (x(s, t), y(s, t), z(s, t))$, two tangent vectors at $f(s_0, t_0)$ can be computed as $\frac{\partial f}{\partial s}(s_0, t_0)$ and $\frac{\partial f}{\partial t}(s_0, t_0)$ where $\frac{\partial f}{\partial s} := \left(\frac{\partial x}{\partial s}, \frac{\partial y}{\partial s}, \frac{\partial z}{\partial s} \right)$. In this example,

$$x(s, t) = L_x \cdot s - (L_x/2)$$

$$y(s, t) = L_y \cdot t - (L_y/2)$$

$$z(s, t) = S \cdot \text{texture}(s, t)$$

- * Examples of smooth parametric surfaces:
 - Bézier surface (demo1, demo2, demo3)
 - NURBS surface: supported by Blender
- 3. Obviously, we can differentiate $x(s, t)$ and $y(s, t)$. But, we cannot differentiate $z(s, t)$ since the texture image is a discrete function. Instead we can approximate it with finite differences, e.g., central difference.:

$$\frac{dg}{du}(u) = \frac{g(u + \delta) - g(u - \delta)}{2\delta}$$

where δ is a small positive number. Note that, to make this work well, you need to set the `gl.TEXTURE_MIN_FILTER` to `gl.LINEAR`.

- 4. Then we can obtain the normal vector by taking cross product of the two tangent vectors. (followed by normalization)
- * GLSL provides the cross function for cross product.

Lighting/Shading You need to apply lighting effects both for the chooper (body + rotor) and the terrain.

- Put a fixed light (with appropriate properties) in the scene. Don't make the fixed light too bright. Otherwise the effect by the glowing bullets may not be visible.
- You can implement either Phong reflection model or Blinn-Phong reflection model.
- You can choose either Phong interpolation or Gouraud interpolation.
- You should compute lighting for all the (active) multiple lights in the scene. (including the glowing bullets below.)

Glowing bullets Your chopper has a weapon. If you press the Spacebar, a glowing bullet is shot forward.

- Each bullet acts as a point light source and the chopper and the terrain should be lighted accordingly.
- Each bullet traverses the scene being affected by the gravity. Let the initial position and velocity of the bullet is (x_0, y_0, z_0) and (v_{0x}, v_{0y}, v_{0z}) , respectively. If we ignore all the complicated air resistance and everything, our differential equations are

$$\begin{aligned}\frac{d^2x}{dt^2}(t) &= 0 \\ \frac{d^2y}{dt^2}(t) &= 0 \\ \frac{d^2z}{dt^2}(t) &= -g\end{aligned}$$

where g is the gravitational constant. (You can set g to any value you want to make the animation looks ok.) In other words, the velocity doesn't change along the x - and y -axes, but changes along the z -axis due to the gravity. To implement this, you can use a simple numerical method, e.g., the explicit Euler method. (Details will be explained in the lecture.)

- To make things simple, limit the number of maximum active bullets in the scene. (9 in the demo video)
- The bullet should "die" if its z value is too low. (e.g., if $z < 0$)

2 NOTE

- If you cannot implement the project using WebGL2, you can use Three.js. But you may have some penalty instead.
- To implement the movement of the chopper, it might help to implement it using Three.js first.
- Compress the *.js files and the *.html in one zip file.
- You need to submit a README file (plain ascii file) which describes all the requirements you (1) succeeded to implement and (2) failed to implement.