CSE 167 Fall 2019

Discussion 10

Final Project

- Project specifications HERE
- DUE Thurs Dec 12 at 3pm
  - Video Presentation - CSE 1242
  - Grading - CSE Basement 260/270
  - NO late submissions
- Graded based on:
  - Blog
  - Video
  - Application

Blog & Video

- Create a blog with (at least) 3 entries documenting your work on this project
  - Blog #1 DUE last Wednesday ...
  - Blog #2 DUE Wednesday Week 10 (12/4) by 11:59 pm
  - Blog #3 DUE Wednesday Finals Week (12/11) by 11:59 pm
- Create a video of your application

Blog #2

- DUE this Wednesday!
- Needs to include (at a minimum)
  - Updates to your team (if any)
  - Updates on your planned features
  - Description of what was implemented during the past week
  - Screenshot of current state of your application
- Note:
  - Blog #3 requirements are the same as Blog #2

Video

- Video/screen capture your application
  - 1 min (1:20 max)
  - Upload to Youtube and add to the playlist posted on Canvas
  - Make sure the video shows the features that you implemented
- These videos will be shown during the first hour of the final
  - Due by 3pm the day of the Final

Application

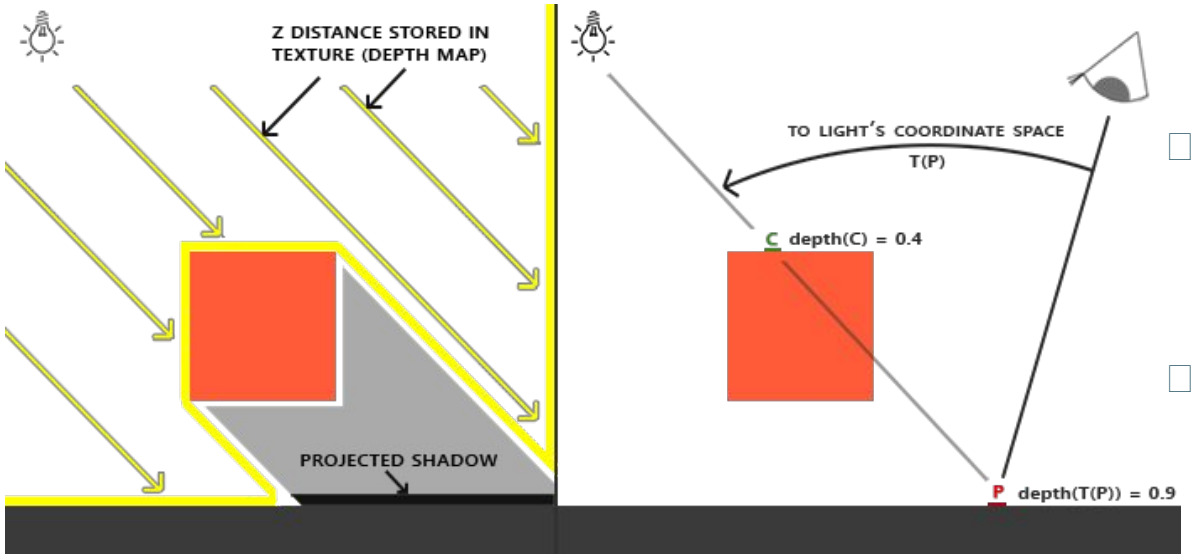
- Implement 1 medium or hard feature for each team member
- Features we are going to cover today:
  - Shadow Mapping
  - Procedurally Generated Terrain

Shadow Mapping

Shadow Mapping

- Tutorial: https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

Shadow Mapping



- First pass: create depth map (yellow region)
- Second pass:
  - For each point P, transform it to light coordinate space with T(P)
  - Compare the depth of T(P) with the depth of corresponding point C in depth map
  - If depth(T(P)) > depth(C), then P is in shadow

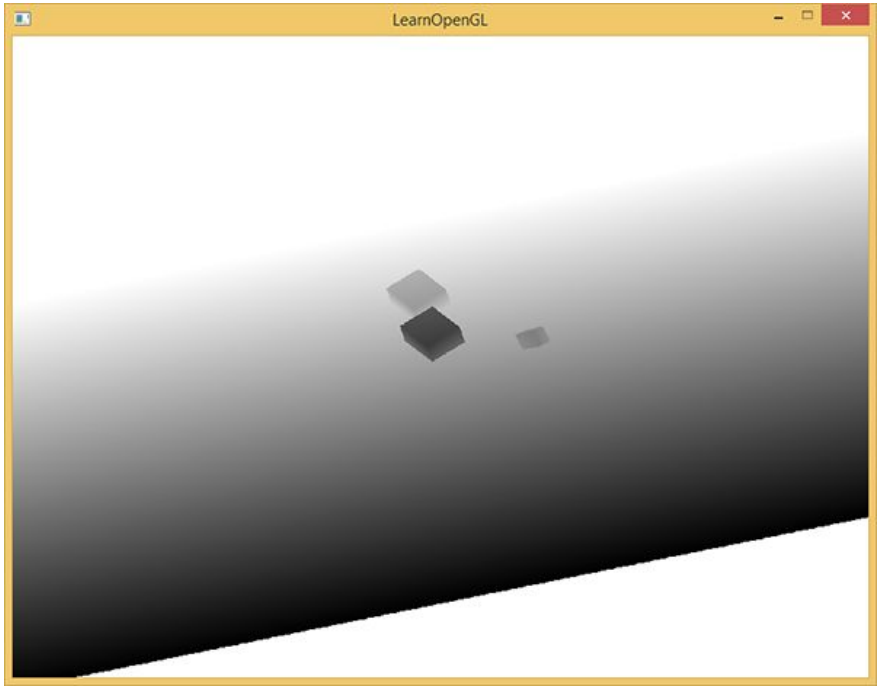
```
//create frame buffer to render depth map
unsigned int depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);

//create a texture as depth buffer
unsigned int depthMap;
glGenTextures(1, &depthMap);
glBindTexture(GL_TEXTURE_2D, depthMap);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_WIDTH, SHADOW_HEIGHT,
    0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);

// attach depth texture as FBO's depth buffer
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, depthMap, 0);
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Render Depth Map

- Calculate light space matrix
  - E.g. a directional light pointing to origin
  - glm::mat4 lightProjection = glm::ortho(...);
  - lightView = glm::lookAt(lightPos, glm::vec3(0), glm::vec3(0, 1, 0));
  - lightSpaceMatrix = lightProjection * lightView;
- Bind frame buffer
  - glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
- Render scene
  - If you don't want shadow for everything (better performance), just render the objects that have shadow

Render Depth Map

- Shader examples can be found on tutorial
- Depth map will look like this
  - Remember to enable the toggle key to show depth map



Render Scene

- Finally in your normal fragment shaders
  - uniform sampler2D shadowMap;
  - float shadow = ShadowCalculation(FragPosLightSpace);
  - fragColor = vec4(vec3(fragColor) * (1-shadow), 1.0);

```
float ShadowCalculation(vec4 fragPosLightSpace)
{
    // perform perspective divide
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // transform to [0,1] range
    projCoords = projCoords * 0.5 + 0.5;
    // get closest depth value from light's perspective (using [0,1] range fragPosLight as coords)
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    // get depth of current fragment from light's perspective
    float currentDepth = projCoords.z;
    // check whether current frag pos is in shadow
    float shadow = currentDepth > closestDepth  ? 1.0 : 0.0;

    return shadow;
}
```

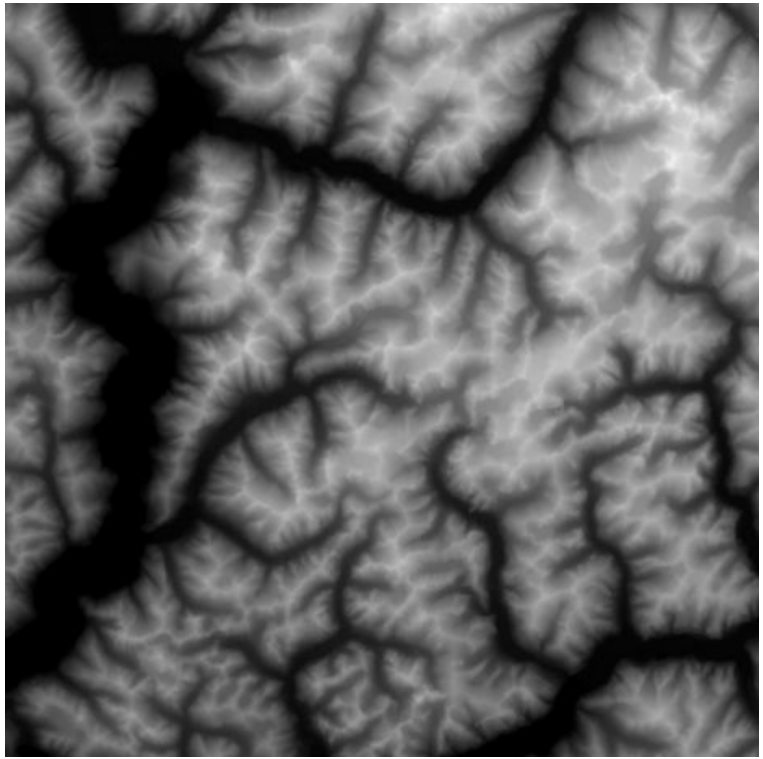
Shadow Mapping

- Improve shadow mapping by fixing the problems
  - Shadow acne
  - Peter panning
  - Over sampling
  - Percentage-closer filtering (PCF)

Procedurally Generated Terrain

Procedurally Generated Terrain

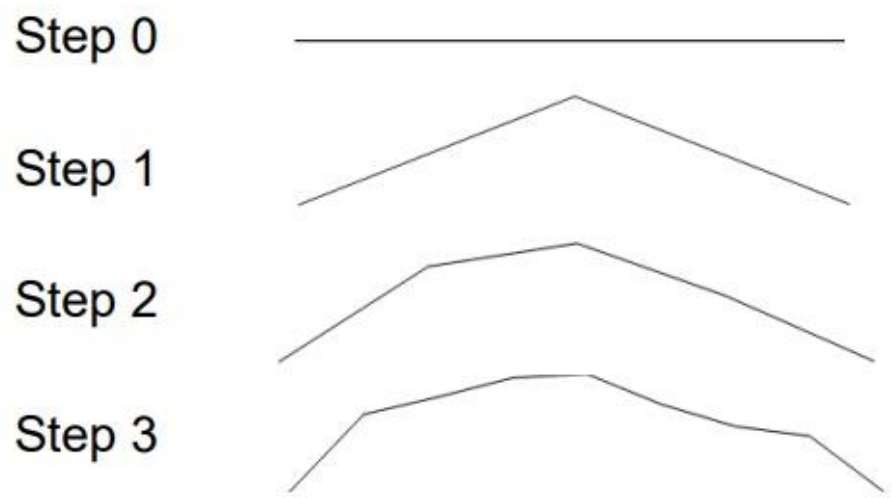
- Construct landscapes from height fields
- Create height field:
  - Midpoint Displacement Algorithm
  - Diamond Square Algorithm
  - Perlin Noise



Midpoint Displacement

1D Midpoint Displacement Alg

```
Start with single horizontal line segment.
Repeat for sufficiently large number of times
{
  Repeat over each line segment in scene
  {
    Find midpoint of line segment.
    Displace midpoint in Y by random amount.
    Reduce range for random numbers.
  }
}
```



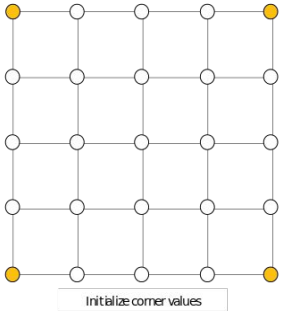
Midpoint Displacement

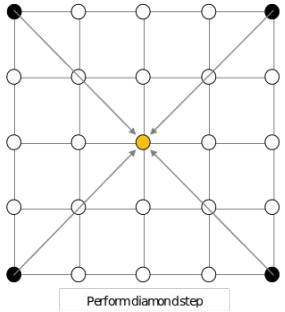
2D Midpoint Displacement Alg

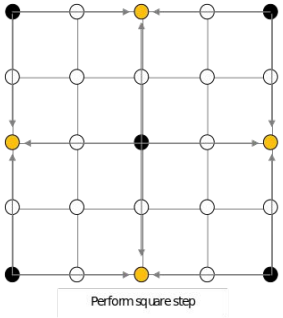
Diamond Square Algorithm

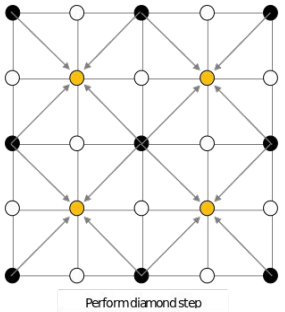
- Diamond Step:
  - For each square in the array, set the midpoint of that square to be the average of the 4 corner points + random value
- Square Step:
  - For each diamond in the array, set the midpoint of that diamond to be the average of the 4 corner points + random value
- At each iteration the magnitude of the random value should be reduced

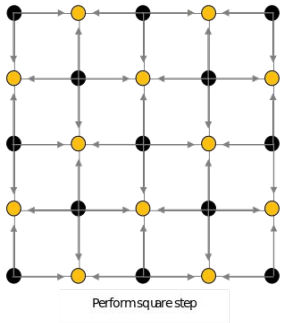
Diamond Square Algorithm





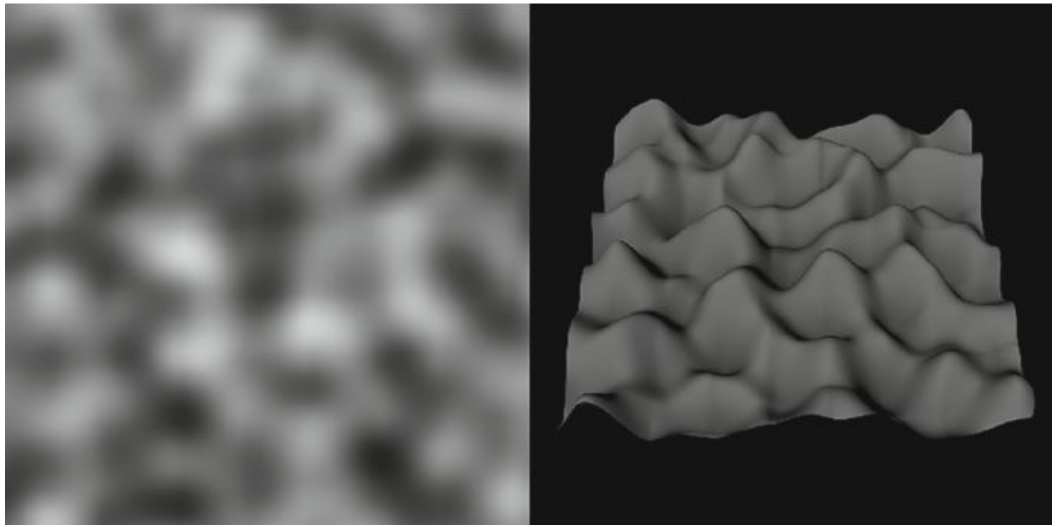






Perlin Noise

- Use type of randomness to generate the height map

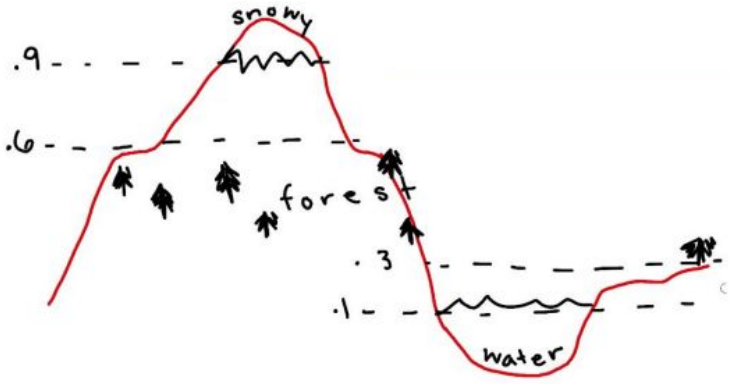


Procedurally Generated Terrain

- Need to generate your terrain information
  - Position coordinates
    - height/y comes from the generated height field
  - Normal, and index information
    - For the triangles that make up your terrain
  - Texture coordinates
    - If you are using textures for your final image

Procedurally Generated Terrain

- Texturing terrain:
  - Define different textures/colors based on the elevation
  - Ex:
    - 0.0 - 0.1 = water
    - 0.3 - 0.4 = forrest
    - 0.9 - 1.0 = snow
    - ect.



For Other Features

Check out previous year’s discussion slides for more information

Questions?