Shadow Maps

Advanced Graphics Programming

Casting Shadows

- The shading algorithms presented so far have all assumed that each light will contribute to the final color of each fragment.
- However, in a complex scene with lots of objects, objects will cast shadows on each other and upon themselves.
- If these shadows are omitted from the rendered scene, a great deal of realism can be lost.

Casting Shadows

- The most basic operation of any shadow calculation must be to determine **whether the point being considered has any light hitting it**.
- We must determine whether there is line of sight from the point being shaded to a light
- Therefore, from the light to the point being shaded.
- We determine whether a piece of geometry is visible from a given vantage point — the depth buffer

Casting Shadows

- Shadow mapping is a technique that produces visibility information for a scene by **rendering it from the point of view of a light source**.
- Only the depth information is needed, and so to do this, we can use a **framebuffer object** with **only a depth attachment**.
- After rendering the scene into a depth buffer from the light's perspective, we will be left with a per-pixel distance of the nearest point to the light in the scene.

Shadow Map

- We render the scene from the lights perspective, and store depth info in the framebuffer

Casting Shadows

- We then render the scene again from the cameras perspective.
- We can calculate, for each point, what the distance to the light is and compare that to the distance stored in the depth buffer.
- The information stored in the shadow map is in screenspace which needs to be converted to texture coordinate space.

Casting Shadows

- Once we have this coordinate, we simply read from the depth texture we rendered earlier.
- We compare our calculated depth value against the one stored in the texture.
- If we are not the closest point to the light for that particular texture, we know we are in shadow.

Camera’s Point of View

- Check if the pixel is in shadow or not

Depth Buffer from Light's Perspective

- The Tree hides the pixel so its in shadow

ShadowMap Class

- Create a shadowmap class and pass in the light and camera.
- In the constructor create shadowmap texture and framebuffer
- You will also have a shadowmapStart and ShadowmapEnd function
- Create a getTexture function and return the shadowmap texture.

Shadow Map – init()

```
// depth texture
Gluint depthMapTexture;
glGenTextures(1, &depthMapTexture);
glBindTexture(GL_TEXTURE_2D, depthMapTexture);

glTexImage2D(GL_TEXTURE_2D,
0, //mipmap level
GL_DEPTH_COMPONENT, //internal format
1280, //screen width
720, //screen height
0, //border
GL_DEPTH_COMPONENT, //color format
GL_FLOAT, //data type
NULL);
```

Shadow Map – init()

```
Gluint depthMapFBO; // global variable
glGenFramebuffers(1, &depthMapFBO);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);

// attach depth texture as FBO's depth buffer
glFramebufferTexture2D(GL_FRAMEBUFFER,
    GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, depthMapTexture, 0);

//disable writes to color buffer
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0); // unbind buffer
```

Shadow Map – init()

- Check if framebuffer is built properly

```
GLenum Status =
    glCheckFramebufferStatus(GL_FRAMEBUFFER);

if (Status != GL_FRAMEBUFFER_COMPLETE) {
    printf("FB error, status: 0x%x\n", Status);
}
```

Shadowmap - Start()

```
void ShadowMap::shadowMapStart(){

glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glClear(GL_DEPTH_BUFFER_BIT);
}
```

ShadowMap- End()

```
void ShadowMap::ShadowMapEnd(){

ActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, depthMapTexture);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
}
```

Changes to Object Class

- Since each object that is required to cast a shadow needs to be drawn from the lights perspective first and then from cameras perspective, new function needs to be added.
- Call this function shadowPass in the object.
- You will also required a new shader program to render from the lights perspective.

Shadwmap Pass Funtion

```
glUseProgram(this->shadowMapProgram);

glm::mat4 lightViewMatrix = glm::lookAt(light->getPosition(), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 1.0f));

glm::mat4 lightVPMatrix = camera->getprojectionMatrix() * lightViewMatrix;
```

Shadow Map pass (cont)

GLint vpLoc = glGetUniformLocation(**shadowMapProgram**, "lightVPMatrix");
glUniformMatrix4fv(vpLoc, 1, GL_FALSE, glm::value_ptr(lightVPMatrix));

// model matrix of the current object
GLint modelLoc = glGetUniformLocation(**shadowMapProgram**, "model");
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelMatrix));

// draw the current object
glBindVertexArray(vao);
glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_INT, 0);
glBindVertexArray(0);

Shadow Map program vs

```
#version 330 core
layout (location = 0) in vec3 position;

uniform mat4 lightVPMatrix;
uniform mat4 model;

void main(){
   gl_Position = lightVPMatrix * model * vec4(position, 1.0);
}
```

Shadow Map program fs

```
#version 430 core

void main(){

        // nothing required here

}
```

Changes to render() of object

- While rendering from the cameras perspective the shadowmap texture needs to be passed in.
- We also need to pass in the lightVPMatrix

Render pass vertex shader

```
#version 450 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec2 texCoord;
layout (location = 2) in vec3 normal;

out vec2 TexCoord;
out vec3 Normal;
out vec4 FragPosLightSpace;

uniform mat4 vp;
uniform mat4 model;
uniform mat4 lightVPMatrix;

void main(){
	FragPos = vec3(model* vec4(position, 1.0f));
	Normal = mat3(transpose(inverse(model))) * normal;
	TexCoord = texCoord;
	FragPosLightSpace = lightVPMatrix * vec4(FragPos, 1.0f);

	gl_Position = vp * model *vec4(position, 1.0);
}
```

render pass fragment shader (cont)

```
#version 450 core
in vec2 TexCoord;  in vec3 Normal;
in vec4 FragPosLightSpace;

out vec4 fragColor;

// texture
uniform sampler2D Texture; uniform sampler2D shadowMap;
//lighting
uniform vec3 objectColor; uniform vec3 cameraPos;
uniform vec3 lightPos; uniform vec3 lightColor;
uniform float specularStrength; uniform float ambientStrength;
```

Render pass fragment shader (cont)

// **old** lighting calculation

//vec3 totalColor = (ambient + diffuse + specular + rim) * objectColor;

// **new** lighting calculation with shadow

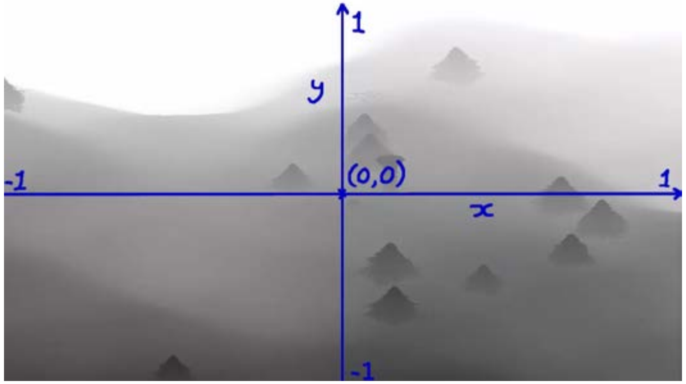
float shadow = ShadowCalculation(FragPosLightSpace);

vec3 totalColor = ambient + ((shadow) * (diffuse + specular + rim));

fragColor = vec4(totalColor, 1.0f) * texture(Texture, TexCoord) ;

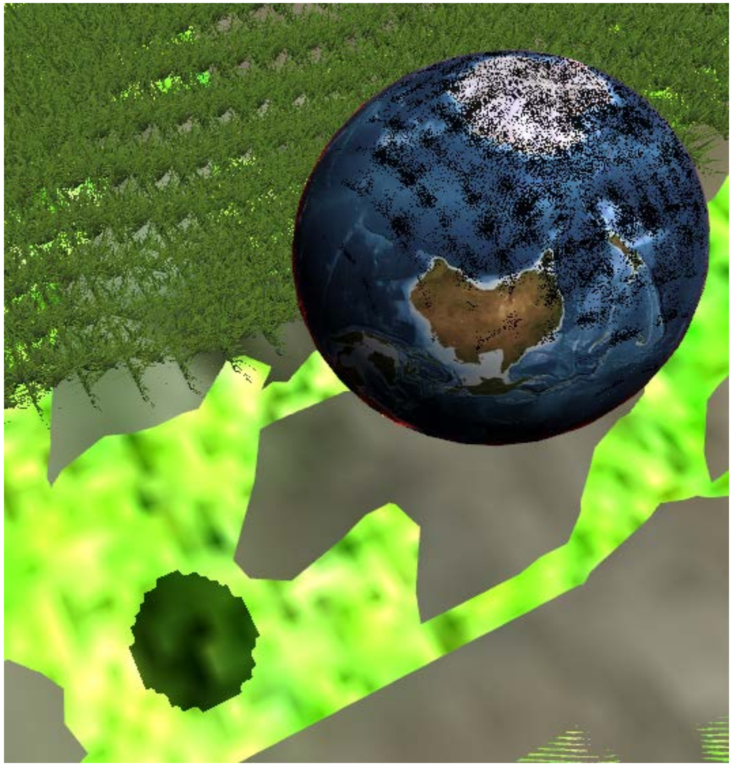
Render pass fragment shader (cont)

```
float ShadowCalculation(vec4 fragPosLightSpace){

// To get NDC [-1, 1] from screenspace
vec3 ndcSpace = fragPosLightSpace.xyz/fragPosLightSpace.w;

// Convert to Tex Coord Space [0,1]
vec3 texCoordSpace =  0.5f * ndcSpace + 0.5f;

float currentDepth = texCoordSpace.z ;

Float closestDepth =
            texture(shadowMap, texCoordSpace.xy).r;


float shadow = currentDepth  < closestDepth ? 1.0 : 0.0;

return shadow
}
```



Shadow Problems

- Shadow Acne and low resolution shadow quality.



Solutions

- Shadow Acne
  - Add a bias to offset the casting of shadows
  - Adjust the value for best solution

```
float bias = .005f;
float currentDepth = texCoordSpace.z - bias;
```

Solutions

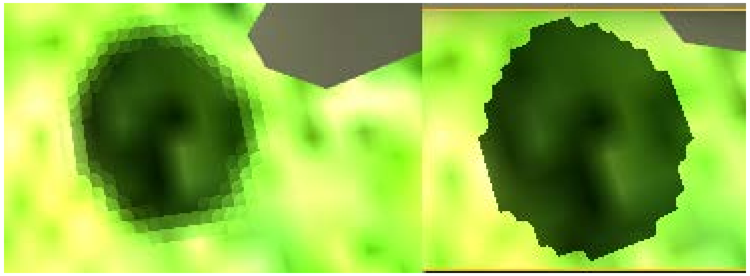
- Shadow Quality
  - Percentage Closer Method
- Keep currentDepth value the same and Comment out

```
closestDepth = texture(shadowMap, shadowCoord.xy/shadowCoord.w).z;
shadow = currentDepth  > closestDepth ? 0.0 : 1.0;
```

- Add following lines

```
vec2 texelSize = 1.0 / textureSize(shadowMap, 0);
for(int x = -1; x <= 1; ++x){
   for(int y = -1; y <= 1; ++y){
   float pcfDepth = texture(shadowMap,
                          texCoordSpace.xy + vec2(x, y) * texelSize).x;
   shadow += currentDepth < pcfDepth ? 1.0 : 0.0;
   }
}

shadow /= 9.0;
```

- Return shadow



Solutions

- Shadow Quality
  - Higher resolution shadow map
  - Use 2K, 4K or 8K resolution
- In shadow map class change texture size
- glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, **4096, 4096,** 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);

Solutions

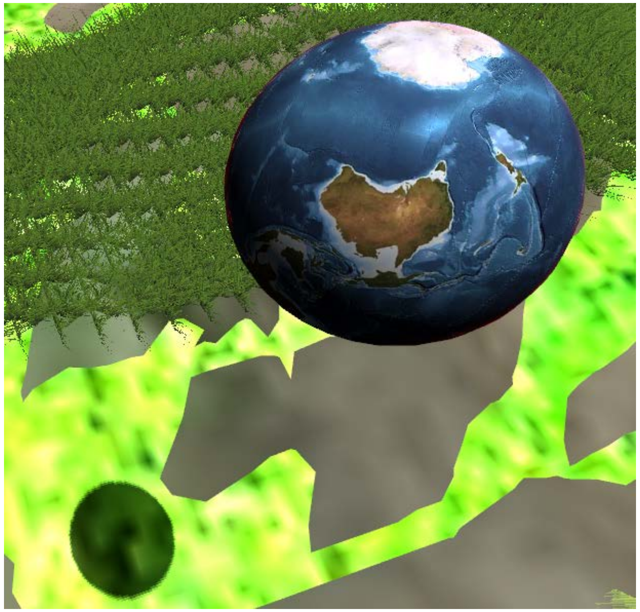
- Change shadowmap start and shadowmap end functions

void ShadowMap::renderFrameBufferStart(){

    glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
    glClear(GL_DEPTH_BUFFER_BIT);
    glViewport(0, 0, 4096,4096);
}

void ShadowMap::renderFrameBufferEnd(){

    glFlush();
    glFinish();
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glViewport(0, 0, Utils::WIDTH, Utils::HEIGHT);
}



Usage

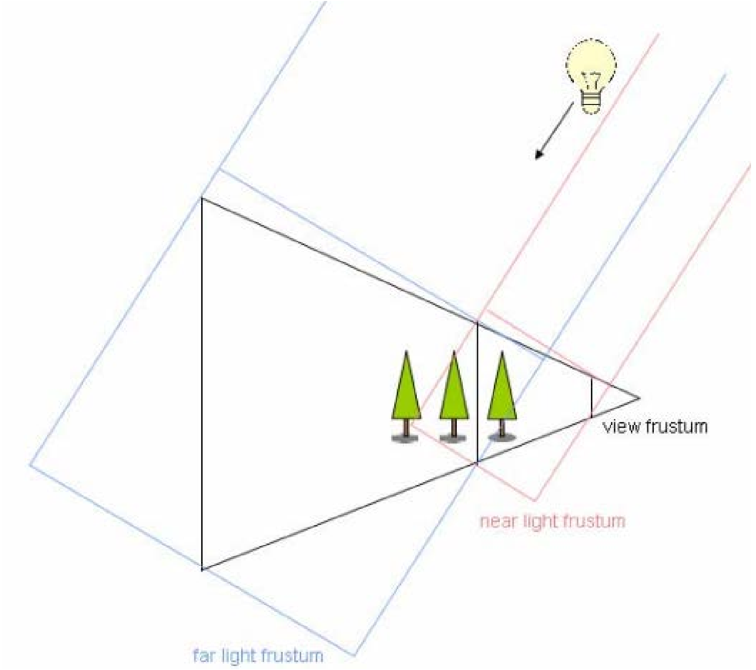
```
//** shadow map start
glCullFace(GL_FRONT);

/////shadow map begin
shadowMap->renderFrameBufferStart();
cube->shadowMapPass();
sphere->shadowMapPass();
shadowMap->renderFrameBufferEnd();

glCullFace(GL_BACK);

cube->renderPass(shadowMap);
sphere->renderPass(shadowMap);
terrain->render(shadowMap->getShadowMapTexture());
```

Final Output

Advanced techniques

- Cascaded Shadow Maps
  - Provides higher resolution of the depth texture near the viewer and lower resolution far away.
  - This is done by splitting the camera view frustum and creating a separate depth-map for each partition.

Advanced techniques

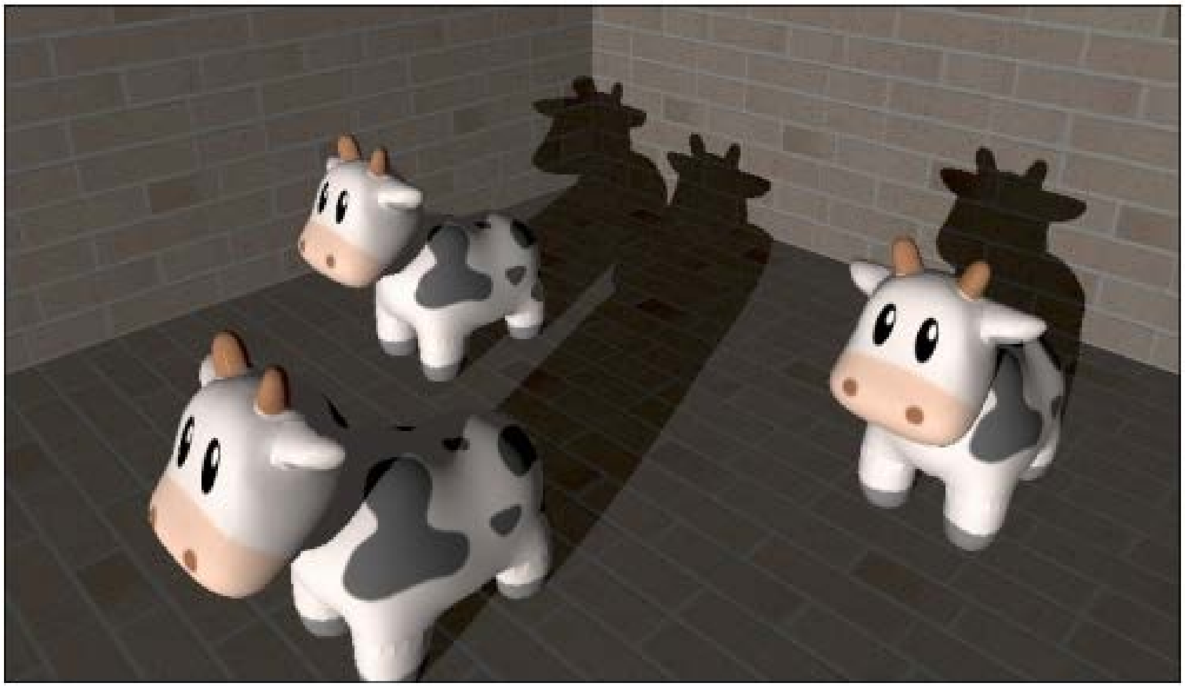


- https://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf

Advanced techniques

- Shadow Volumes
  - Shadows using shadow volumes and the geometry shader
  - You get pixel-perfect hard shadows, without the aliasing artifacts of shadow maps.
  - The shadow volume technique works by making use of the stencil buffer to mask out areas that are in shadow.
  - A shadow volume is the region of space where the light source is occluded by an object.

Advanced techniques



- https://software.intel.com/sites/default/files/salvi_avsm_egsr2010.pdf

Excercise

- Implement shadow map for objects in a 3D scene.
- Add corrective measure to remove shadow acne.
- Increase shadow quality by adding PCF algorithm and increase Shadow map texture size.