Jack Belford
V00829017

Assignment #2 – Real Time Rendering

In this assignment, various features for the provided OpenGL project were implemented.

These include:

1. Transforming vertices to clip space by using a ModelViewProjection matrix in the vertex shader
2. Phong shading in the fragment shader
3. Hierarchy of transformations
4. Shadow mapping
5. Rendering the sponza scene
6. Skybox

1 & 2:

```
void main()
{
    gl_Position = ModelViewProjection * Position;
    fragment_texcoord = TexCoord;
    fragment_normal   = normalize(Normal_ModelWorld * Normal);
    fragment_position = (ModelWorld * Position).xyz;
    shadowMapCoord = LightMatrix * ModelWorld * Position;
}
```

Figure 1: Vertex Shader

In the provided code the ModelViewProjection matrix was already calculated and sent to the vertex shader so the transformation in part 1 was as simple as multiplying it with the vertex position vector. For Phong shading the calculations are performed in the fragment shader so I sent the texture coordinates, normal, position in world space.

Jack Belford
V00829017

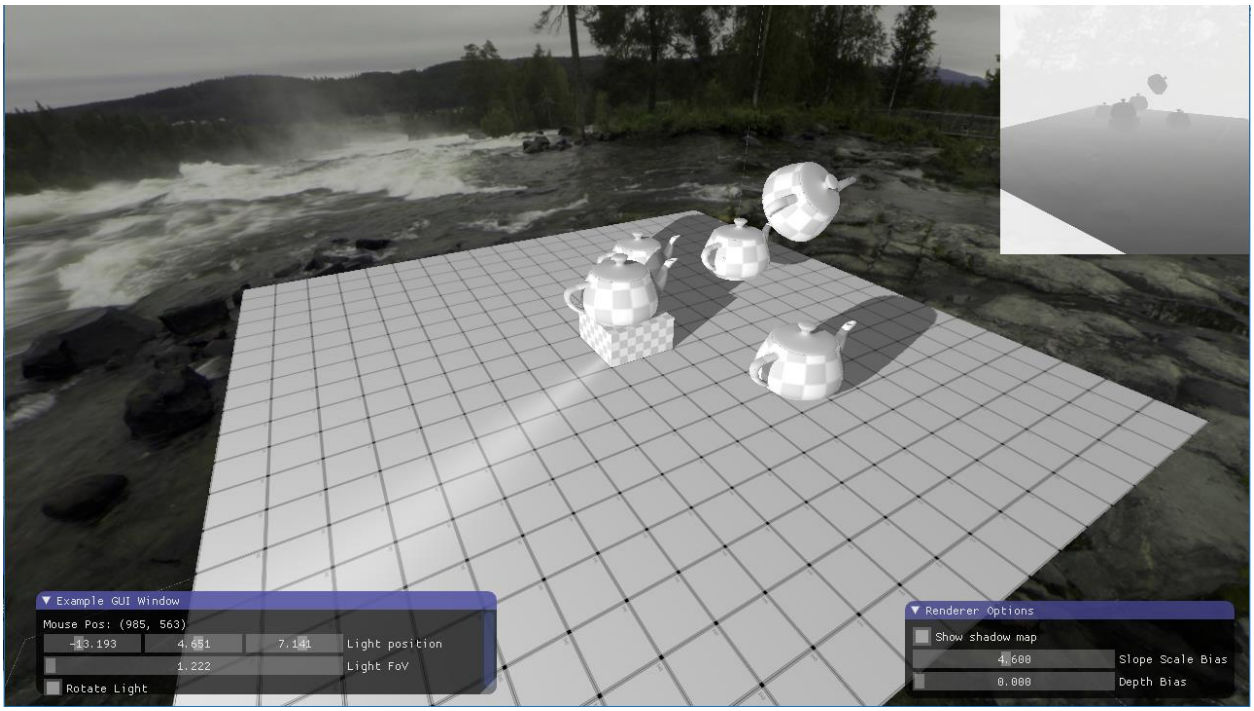
```
void main()
{
    vec3 diffuseMap;
    if (HasDiffuseMap != 0)
        diffuseMap = texture(DiffuseMap,fragment_texcoord).rgb;
    else
        diffuseMap = vec3(1.0);

    vec3 v = normalize(-fragment_position);
    vec3 l = normalize(LightPos - fragment_position);
    vec3 h = normalize(v + l);

    vec3 amb   = Ambient * 0.5;
    vec3 diff  = Diffuse * max(dot(fragment_normal, l), 0.0);
    vec3 spec  = Specular * pow(max(dot(fragment_normal, h), 0.0), Shininess);
    vec3 color = diffuseMap * (amb + diff) + spec;

    float visibility = 1.0f;
    if (IgnoreShadow != 1) {
        visibility = 0.3 + 0.7 * textureProj(ShadowMap, shadowMapCoord);
    }

    FragColor = vec4(color * visibility, 1.0);
}
```

Figure 2: Fragment Shader

In this calculation for Phone shading I used the half vector. Originally the light was set at the Camera position but it later felt more appropriate to make this the actual Light position that was also responsible for shadow mapping. I also didn't like how bright the ambient light made the objects so I halved it (Ambient * 0.5). Max() is used to get rid of negative values. I also used a slightly different form for my Phong calculation. I found that "diffuseMap * (amb + diff) + spec" looked better than "amb + diffuseMap * (diff + spec)" or "amb + diffuseMap*diff + spec" so I just stuck to that.

Jack Belford
V00829017

The result:



* Notice the stronger intensity on the floor traveling from the light to origin (It's target)

3:

```
struct Transform
{
    glm::vec3 Scale;
    glm::vec3 RotationOrigin;
    glm::quat Rotation;
    glm::vec3 Translation;
    glm::quat OriginRotation;
    glm::vec3 axis;
    float speed;
    int32_t parentID;
};
```

Figure 4: Transform object changes

```
uint32_t mainParentID;
uint32_t secondParent;
// place a teapot on top of the cube
{
    uint32_t newInstanceID;
    AddMeshInstance(mScene, loadedMeshID, &newInstanceID, 0);
    mainParentID = scene->Instances[newInstanceID].TransformID;
    scene->Transforms[mainParentID].Translation += glm::vec3(0.0f, 2.0f, 0.0f);
    scene->Transforms[mainParentID].parentID = -1;
    scene->Transforms[mainParentID].speed = -1;
}

// place a teapot in mid air
{
    uint32_t newInstanceID;
    AddMeshInstance(mScene, loadedMeshID, &newInstanceID, 0);
    secondParent = scene->Instances[newInstanceID].TransformID;
    scene->Transforms[secondParent].Translation += glm::vec3(5.0f, 0.0f, 0.0f);
    scene->Transforms[secondParent].parentID = mainParentID;
    scene->Transforms[secondParent].speed = 3;
    scene->Transforms[secondParent].axis = glm::vec3(0, 1, 0);
}
```

Figure 3: Changes to simulation.cpp

Jack Belford
V00829017

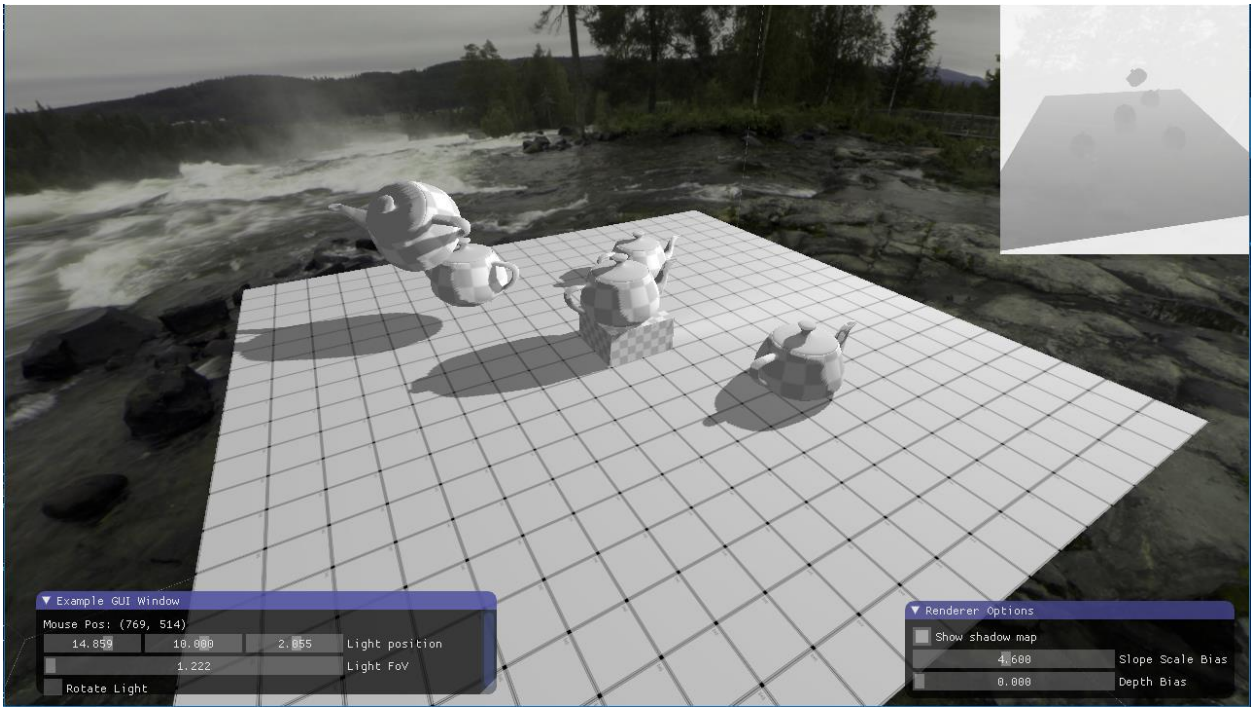
```
// Here is where I implement the hierarchical transforms
glm::mat4 modelWorld;
glm::mat3 normal_ModelWorld;

while (1)
{
    modelWorld = translate(-transform.RotationOrigin) * modelWorld;
    modelWorld = mat4_cast(transform.Rotation) * modelWorld;
    modelWorld = translate(transform.RotationOrigin) * modelWorld;
    modelWorld = scale(transform.Scale) * modelWorld;
    modelWorld = translate(transform.Translation) * modelWorld;

    normal_ModelWorld = mat3_cast(transform.Rotation) * normal_ModelWorld;

    if (transform.speed != -1) {
        float deltaAngle = deltaTime* transform.speed * M_PI * 15.0f / 180.0f; // 15 degrees/second
        transform.OriginRotation = glm::rotate(transform.OriginRotation, deltaAngle, transform.axis);
        modelWorld = mat4_cast(transform.OriginRotation) * modelWorld;
        normal_ModelWorld = mat3_cast(transform.OriginRotation) * normal_ModelWorld;
    }

    normal_ModelWorld = glm::mat3(scale(1.0f / transform.Scale)) * normal_ModelWorld;

    if (transform.parentID != -1) {
        transform = mScene->Transforms[transform.parentID];
    }
    else break;
}

glm::mat4 modelViewProjection = worldProjection * modelWorld;
```

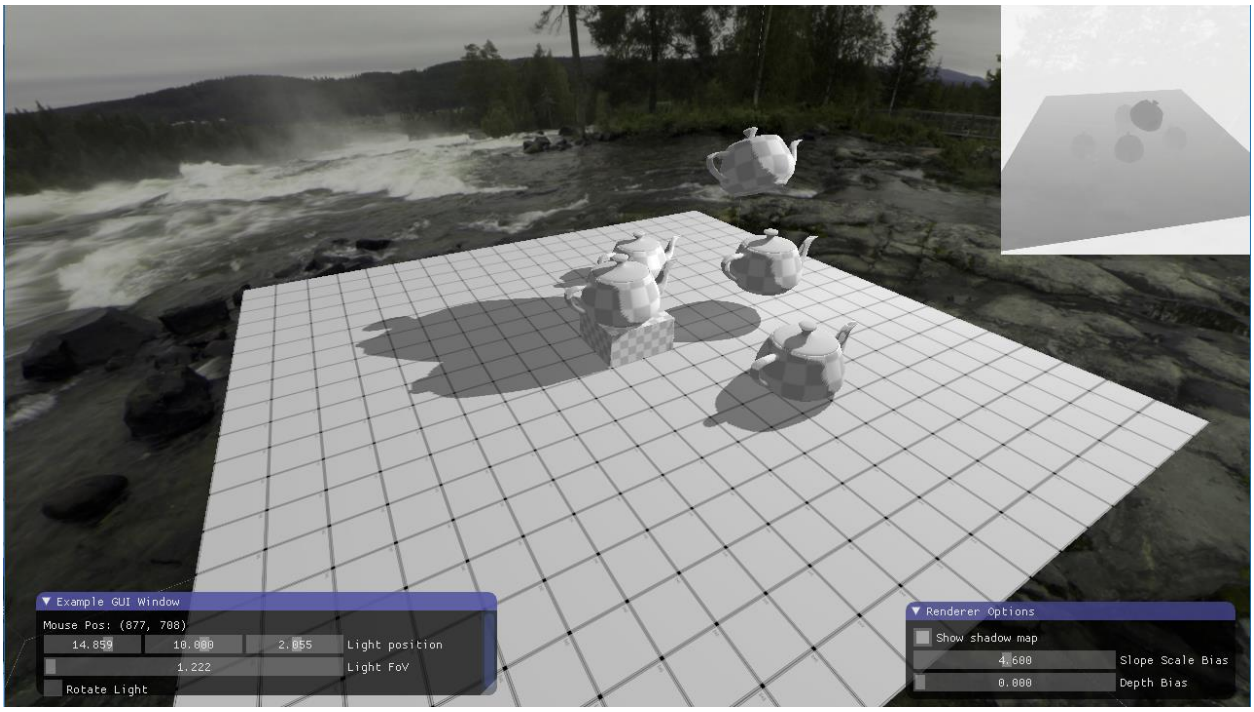
Figure 5: Performing transforms

For the hierarchical transforms I changed the Transform object as in Figure 4. I added a parentID which indicates the ID of the Transform it is relative to. I also added “speed” which indicates if the object is rotating about its parent. OriginRotation is a new quat to keep track of this rotation (Since Rotation is for rotating the object in place). The Axis vector is self-explanatory. In simulation.cpp I changed the initialization of the objects to set their parentID’s as well as speed. If parentID is -1 then it is not relative to any other object. If speed is set then I also set the axis. So based on the simulation.cpp I am expecting there to be a teapot offset by 5.0 units on the x-axis to the teapot that is on top of the cube. This new teapot will also be rotating (orbiting) around the teapot on the cube. In renderer.cpp I perform the transformations as in Figure 5. The modelWorld and normal_ModelWorld vectors are now created by traversing parentID’s.

Jack Belford
V00829017

The result:





*I also made another teapot that rotates about the x-axis around the teapot already rotating about the middle teapot. It is hard to show this operation via pictures so I recommend running the code for yourself.

Jack Belford
V00829017

4:

```
struct Light
{
    glm::vec3 Position;
    glm::vec3 Direction;
    glm::vec3 Up;

    float FovY;
};
```

```
Camera MainCamera;
Light MainLight;
```

Figure 6: Edit the scene to have MainLight

Figure 7: Light object

```
Light mainLight;
mainLight.Position = glm::vec3(-15.0f, 10.0f, 0.0f);
mainLight.Direction = normalize(glm::vec3(0.0f, 0.0f, 0.0f) - mainLight.Position);
mainLight.Up = glm::vec3(0.0f, 1.0f, 0.0f);
mainCamera.FovY = glm::radians(70.0f);
mScene->MainLight = mainLight;
```

Figure 8: Init Light in Simulation.cpp

```
Light& light = mScene->MainLight;
if (ImGui::Begin("Light Options"))
{
    ImGui::SliderFloat3("Light position", value_ptr(light.Position), -40.0f, 40.0f);
    ImGui::SliderFloat("Light FoV", &light.FovY, M_PI * 20.0f/180.0f, M_PI);
    ImGui::Checkbox("Rotate Light", &rotateLight);
    ImGui::SliderFloat("Rotate Speed", &rotateSpeed, -10.0f, 10.0f);
}
ImGui::End();
```

Figure 9: Provide options for manipulating the light position

```
kShadowMapResolution = 1024;
mShadowSlopeScaleBias = 4.6f;
mShadowDepthBias = 0.0f;
glGenTextures(1, &mShadowDepthTO);
glBindTexture(GL_TEXTURE_2D, mShadowDepthTO);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, kShadowMapResolution,
    kShadowMapResolution, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE, GL_COMPARE_REF_TO_TEXTURE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
const float kShadowBorderDepth[] = { 1.0f, 1.0f, 1.0f, 1.0f };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, kShadowBorderDepth);
glBindTexture(GL_TEXTURE_2D, 0);

glGenFramebuffers(1, &mShadowFBO);
glBindFramebuffer(GL_FRAMEBUFFER, mShadowFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, mShadowDepthTO, 0);
glBindFramebuffer(GL_FRAMEBUFFER, 0);

mShadowSP = mShaders.AddProgramFromExts({ "shadow.vert", "shadow.frag" });
```

Figure 10: Create the shadowmap texture and attaching to framebuffer

Jack Belford
V00829017

```
// render shadow map
if (*mShadowSP)
{
    glUseProgram(*mShadowSP);
    glBindFramebuffer(GL_FRAMEBUFFER, mShadowFBO);
    glClearDepth(1.0f);
    glClear(GL_DEPTH_BUFFER_BIT);
    glViewport(0, 0, kShadowMapResolution, kShadowMapResolution);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_POLYGON_OFFSET_FILL);
    glPolygonOffset(mShadowSlopeScaleBias, mShadowDepthBias);

    GLint SCENE_MODELVIEWPROJECTION_UNIFORM_LOCATION = glGetUniformLocation(*mShadowSP, "ModelViewProjection");

    const Light& mainLight = mScene->MainLight;
    glm::vec3 lightPos = mainLight.Position;
    glm::vec3 lightUp = mainLight.Up;
    glm::mat4 worldView = glm::lookAt(lightPos, lightPos + mainLight.Direction, lightUp);
    glm::mat4 viewProjection = glm::perspective(mainLight.FovY, 1.0f, 0.01f, 100.0f);
    glm::mat4 worldProjection = viewProjection * worldView;

    for (uint32_t instanceID : mScene->Instances)
    {
        const Instance* instance = &mScene->Instances[instanceID];
```

Figure 11: Create a render shadow pass

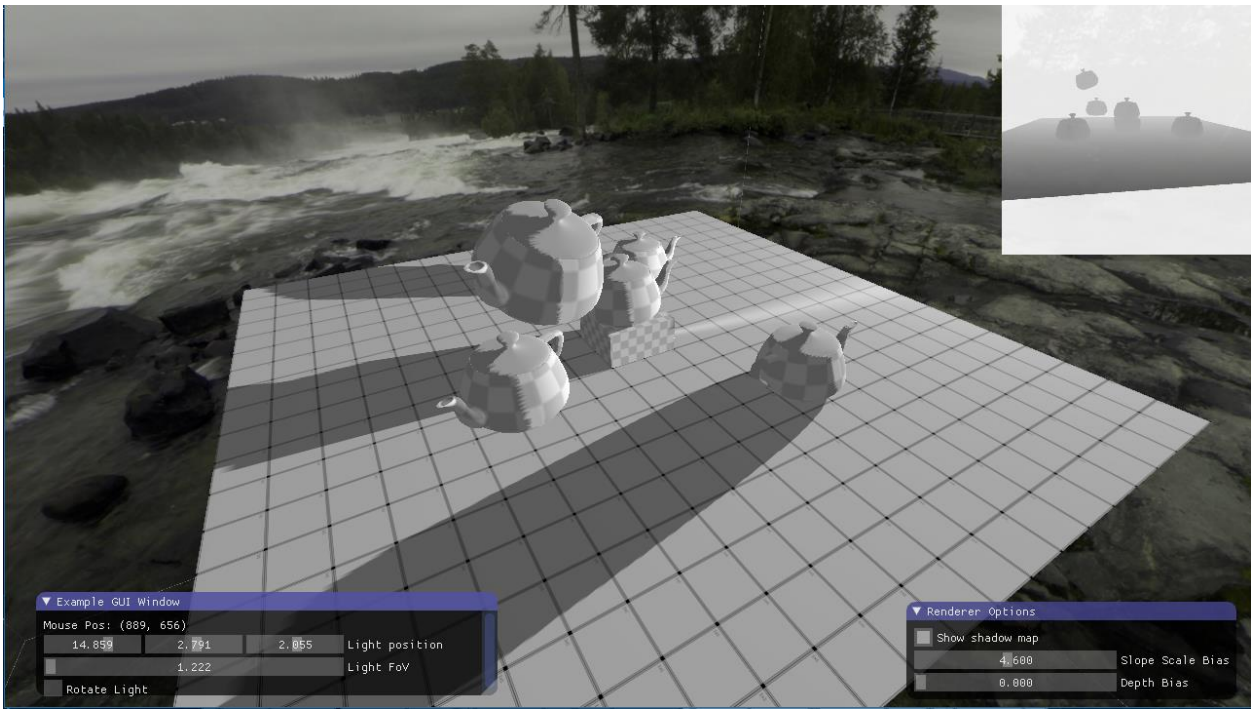
```
const Light& mainLight = mScene->MainLight;
glm::vec3 lightPos = mainLight.Position;
glm::vec3 lightUp = mainLight.Up;
glm::mat4 lWorldView = glm::lookAt(lightPos, lightPos + mainLight.Direction, lightUp);
glm::mat4 lViewProjection = glm::perspective(mainLight.FovY, 1.0f, 0.01f, 100.0f);
glm::mat4 lWorldProjection = lViewProjection * lWorldView;
glm::mat4 lightOffsetMatrix = glm::mat4(
    0.5f, 0.0f, 0.0f, 0.0f,
    0.0f, 0.5f, 0.0f, 0.0f,
    0.0f, 0.0f, 0.5f, 0.0f,
    0.5f, 0.5f, 0.5f, 1.0f);
glm::mat4 lightMatrix = lightOffsetMatrix * lWorldProjection;
glProgramUniformMatrix4fv(*mSceneSP, SCENE_LIGHTMATRIX_UNIFORM_LOCATION, 1, GL_FALSE, value_ptr(lightMatrix));

glActiveTexture(GL_TEXTURE0 + SCENE_SHADOW_MAP_TEXTURE_BINDING);
glBindTexture(GL_TEXTURE_2D, mShadowDepthTO);
glUniform1i(SCENE_SHADOW_MAP_UNIFORM_LOCATION, SCENE_SHADOW_MAP_TEXTURE_BINDING);

glProgramUniform3fv(*mSceneSP, SCENE_LIGHTPOS_UNIFORM_LOCATION, 1, value_ptr(lightPos));
```

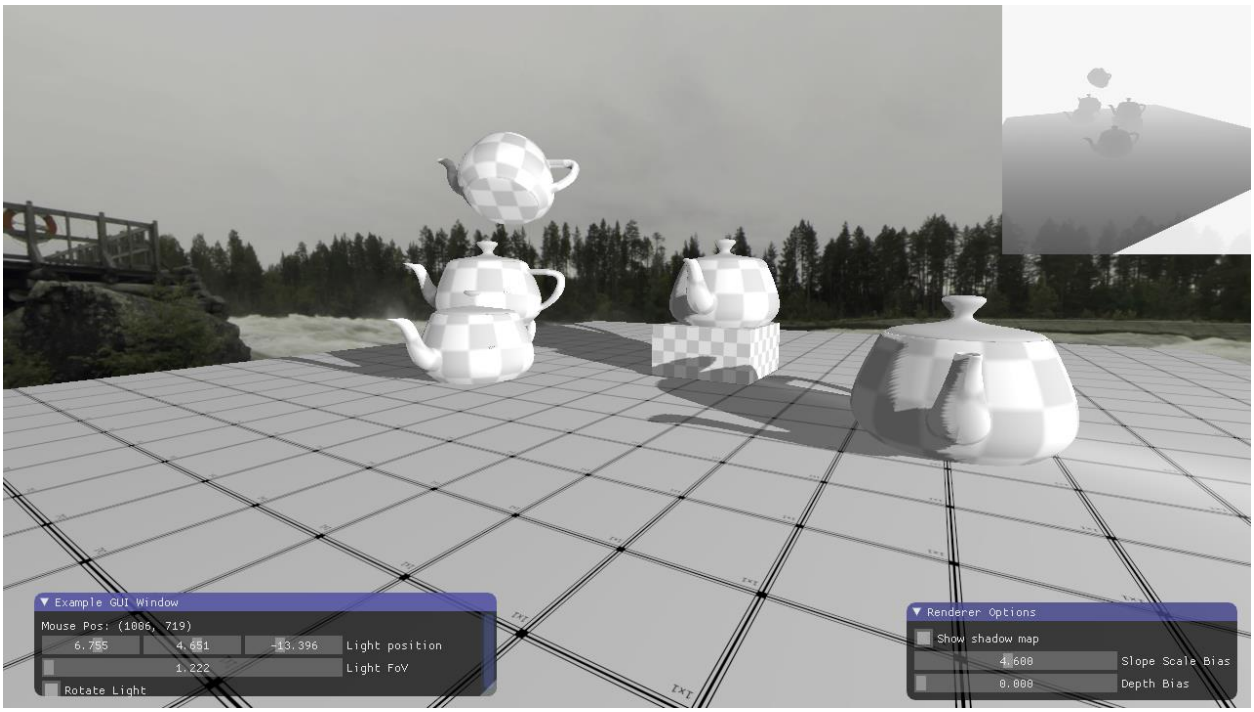
Figure 12: In the scene render pass, pass the shadowmap to vertex and fragment shaders

The previous figures show all the essential changes made to implement shadow mapping. The shadow map rendering pass is basically the same as the main scene rendering pass except that it has much less data to pass to shaders and for each mesh none of the material information is used. Notice in figures 1 & 2 that the shadow map texture coordinates are computed in the vertex shader and then used in the fragment shader to determine whether a point is in shadow. The function textureProj() returns a number between 0 and 1. 0 meaning completely in the shade and 1 meaning completely visible. The values in between 0 and 1 are only really returned at edges. Initially the shadows rendered would be completely black because I didn't edit the value returned by textureProj() before multiplying with the "color". So I decided to set visibility to 0.3 if textureProj() was less than 1. This led to annoying black outlines on all the shadows and that were highly noticeable. It wasn't until after I moved on to Sponza and Skybox that I realized my mistake and used the correct formula: 0.3 + 0.7 * textureProj().

Jack Belford
V00829017

Result:





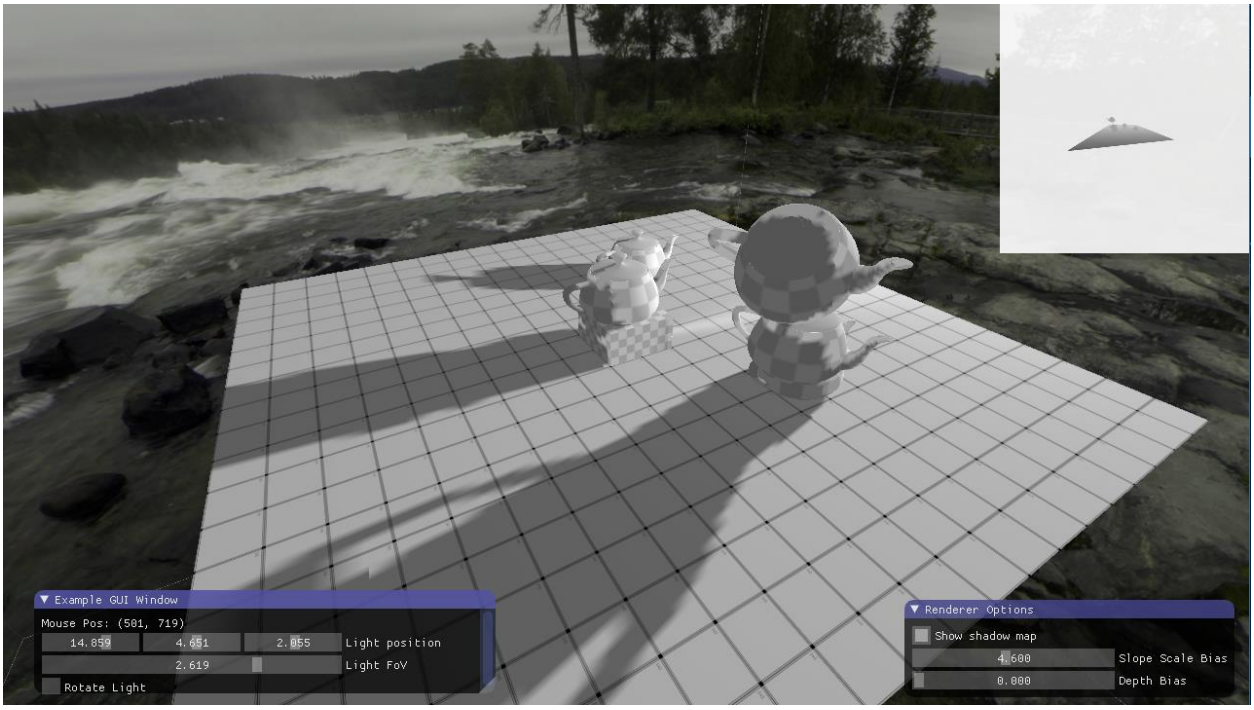


Figure 13: Distortion due to low resolution using a large FoV

Jack Belford
V00829017

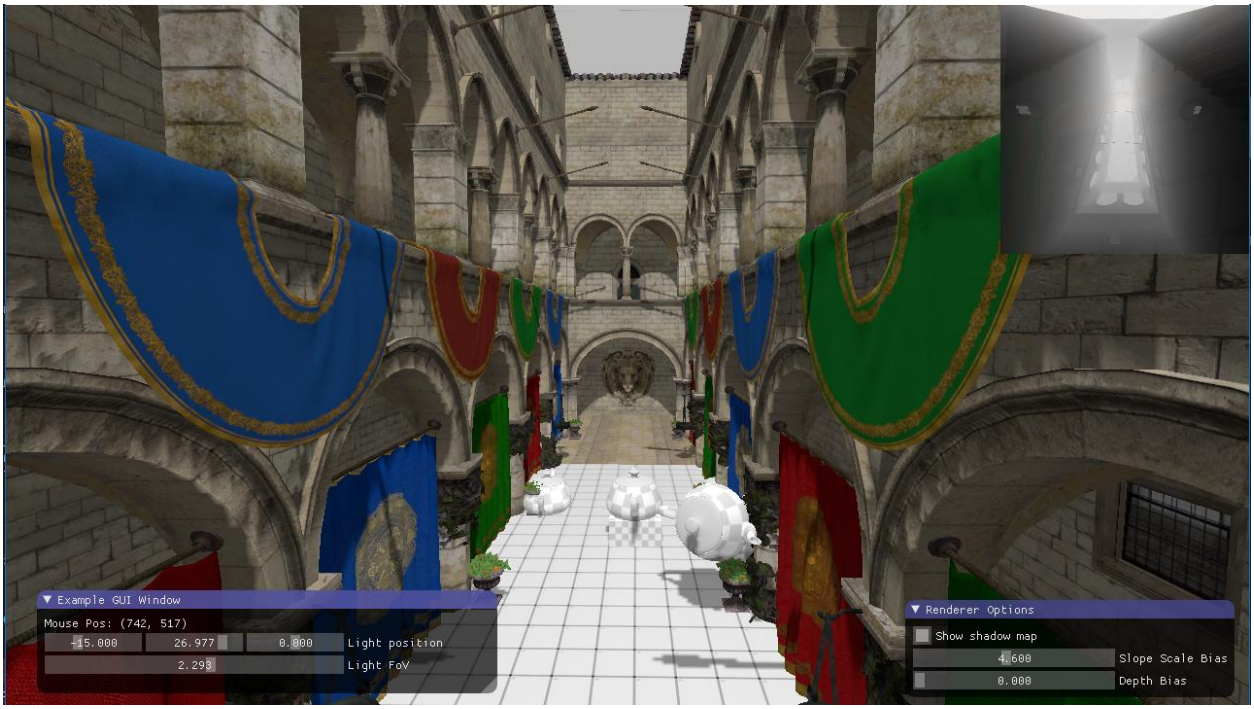
5.

```
loadedMeshIDs.clear();
LoadMeshesFromFile(mScene, "assets/sponza/sponza.obj", &loadedMeshIDs);
for (uint32_t loadedMeshID : loadedMeshIDs)
{
    if (scene->Meshes[loadedMeshID].Name == "sponza_04") continue;
    uint32_t newInstanceID;
    AddMeshInstance(mScene, loadedMeshID, &newInstanceID, 0);
    uint32_t newTransformID = scene->Instances[newInstanceID].TransformID;
    scene->Transforms[newTransformID].Scale = glm::vec3(01.0f / 50.0f);
    scene->Transforms[newTransformID].parentID = -1;
    scene->Transforms[newTransformID].speed = -1;
}
```

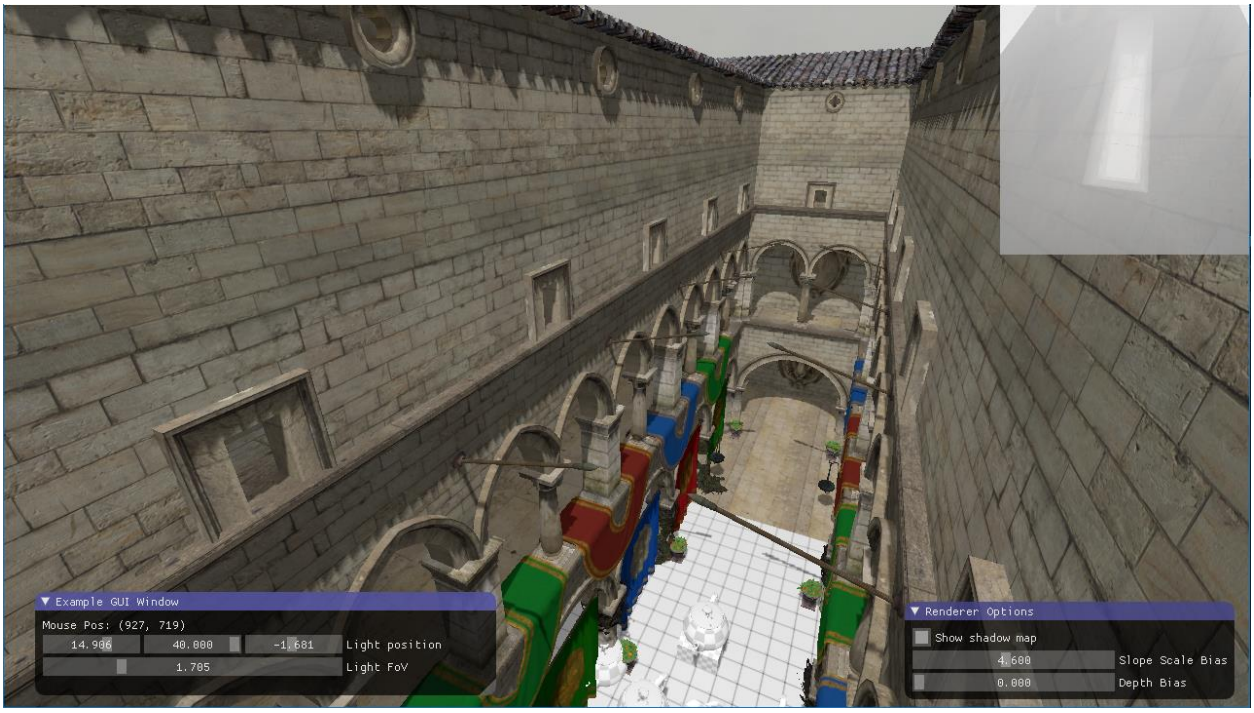
Figure 14: Loading sponza scene in Simulation.cpp

This is pretty much all that was needed once the sponza textures and obj files were downloaded.

Result:



Jack Belford
V00829017



Jack Belford
V00829017

6:

```
g skybox

usemtl up
f -8/-4/-6 -7/-3/-6 -6/-2/-6
f -8/-4/-6 -6/-2/-6 -5/-1/-6

usemtl right
f -8/-4/-5 -4/-3/-5 -3/-2/-5
f -8/-4/-5 -3/-2/-5 -7/-1/-5

usemtl left
f -6/-4/-4 -2/-3/-4 -1/-2/-4
f -6/-4/-4 -1/-2/-4 -5/-1/-4

usemtl far
f -5/-4/-3 -1/-3/-3 -4/-2/-3
f -5/-4/-3 -4/-2/-3 -8/-1/-3

usemtl near
f -7/-4/-2 -3/-3/-2 -2/-2/-2
f -7/-4/-2 -2/-2/-2 -6/-1/-2

usemtl down
f -3/-4/-1 -4/-3/-1 -1/-2/-1
f -3/-4/-1 -1/-2/-1 -2/-1/-1
```

Figure 15: Changes made to cube.obj for skybox.obj

```
newmtl up
  Ns 10.0000
  Ni 1.5000
  Tr 0  0
  illum 2
  Ka 1 1 1
  Kd 1 1 1
  Ks 0.2 0.2 0.2
  Ke 0 0 0
  map_Kd posy.jpg

newmtl down
  Ns 10.0000
  Ni 1.5000
  Tr 0  0
  illum 2
  Ka 1 1 1
  Kd 1 1 1
  Ks 0.2 0.2 0.2
  Ke 0 0 0
  map_Kd negy.jpg

newmtl left
  Ns 10.0000
```

Figure 16: Changes made to default.mtl

For the skybox I created a new folder and copied the files from cube folder but with some changes. I downloaded cube map images labelled posy.jpg, negy.jpg, etc. corresponding to each side. In simulation.cpp I then added this new object.

```
loadedMeshIDs.clear();
LoadMeshesFromFile(mScene, "assets/skybox/skybox.obj", &loadedMeshIDs);
for (uint32_t loadedMeshID : loadedMeshIDs)
{
    uint32_t newInstanceID;
    AddMeshInstance(mScene, loadedMeshID, &newInstanceID, 1);

    // scale up the cube
    uint32_t newTransformID = scene->Instances[newInstanceID].TransformID;
    scene->Transforms[newTransformID].Scale = glm::vec3(100.0f);
    scene->Transforms[newTransformID].parentID = -1;
    scene->Transforms[newTransformID].speed = -1;
}
```

Figure 17: Skybox object init

This renders the skybox but one problem with it is that it still has shadows cast to it. To fix this I edited the Instance object to have an “Ignore” variable. This variable is sent to the fragment shader which will first check that Ignore is not set to 1 before computing visibility. (Figure 2).

```
struct Instance
{
    uint32_t MeshID;
    uint32_t TransformID;
    bool Ignore;
};
```

```
void AddMeshInstance(
    Scene* scene,
    uint32_t meshID,
    uint32_t* newInstanceID,
    int ignore)
{
    Transform newTransform;
    newTransform.Scale = glm::vec3(1.0f);

    uint32_t newTransformID = scene->Transforms.insert(newTransform);

    Instance newInstance;
    newInstance.MeshID = meshID;
    newInstance.TransformID = newTransformID;
    newInstance.Ignore = ignore;
```

Figure 18: AddMeshInstance was changed to store this ignore value

```
glProgramUniform1i(*mSceneSP, SCENE_IGNORE_UNIFORM_LOCATION, instance->Ignore);
glProgramUniformMatrix4fv(*mSceneSP, SCENE_MODELWORLD_UNIFORM_LOCATION, 1, GL_FALSE, value_ptr(modelWorld));
glProgramUniformMatrix3fv(*mSceneSP, SCENE_NORMAL_MODELWORLD_UNIFORM_LOCATION, 1, GL_FALSE, value_ptr(normal_ModelWorld));
glProgramUniformMatrix4fv(*mSceneSP, SCENE_MODELVIEWPROJECTION_UNIFORM_LOCATION, 1, GL_FALSE, value_ptr(modelViewProjection));
```

Figure 19: Passing ignore to fragment shader

Result: *Shown in previous non-sponza images*

Final notes:

My code is currently setup to render the scene without Sponza but this can be changed by un-commenting out the following code found in simulation.cpp:

```
/*
loadedMeshIDs.clear();
LoadMeshesFromFile(mScene, "assets/sponza/sponza.obj", &loadedMeshIDs);
for (uint32_t loadedMeshID : loadedMeshIDs)
{
    if (scene->Meshes[loadedMeshID].Name == "sponza_04") continue;
    uint32_t newInstanceID;
    AddMeshInstance(mScene, loadedMeshID, &newInstanceID, 0);
    uint32_t newTransformID = scene->Instances[newInstanceID].TransformID;
    scene->Transforms[newTransformID].Scale = glm::vec3(01.0f / 50.0f);
    scene->Transforms[newTransformID].parentID = -1;
    scene->Transforms[newTransformID].speed = -1;
} */
```