CSC305_A2 Report

V00825808 Xinyue Liu

- In the vertex shader, transform vertices to clip space using the ModelViewProjection matrix.

  This part requires our program do a transformation from model space to clip space, because APIs only care about clip space. As the instruction from lab5.pdf, we can use gl_Position to output clip space coordinate. The “model view projection” matrix is used for transforming model->clip by using matrix multiplication.

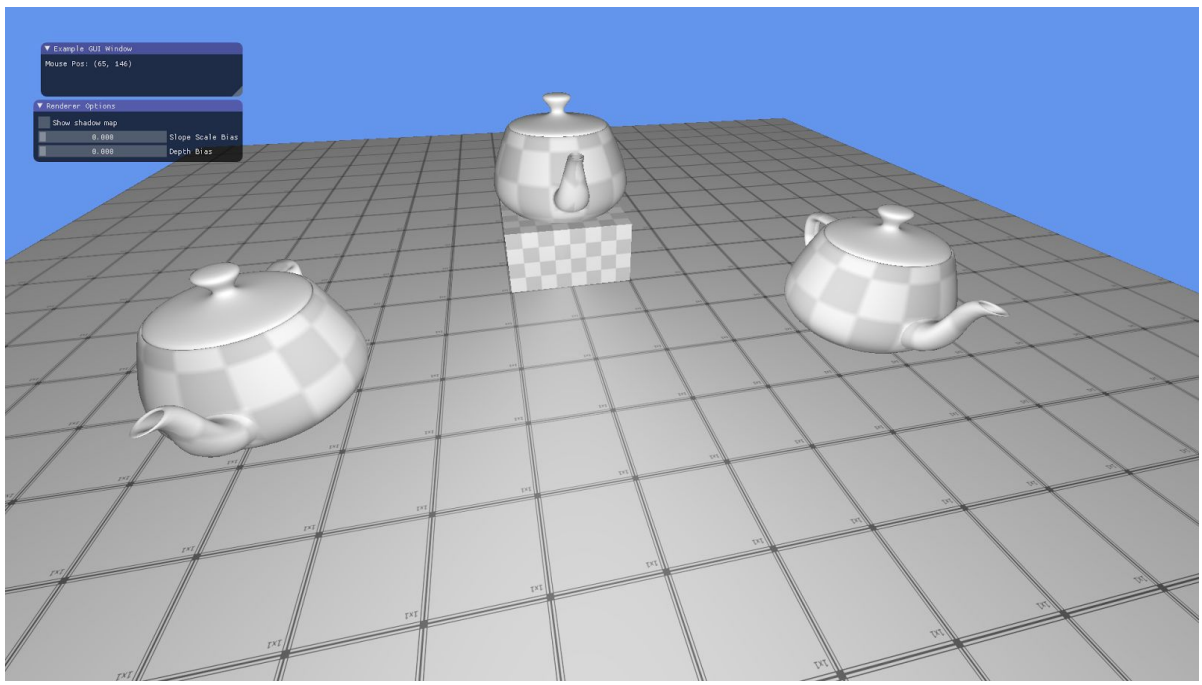
- In the pixel shader. implement Phong shading for a point light placed at the camera’s position.

  This part is to implement the Blinn–Phong shading model in order to determine the diffuse and specular light from a point light source placed at the camera’s position. Here is the calculation for Phong shading: H = (L+V)/|L+V| (H: halfway vector L:light-source V: viewer). Apply this calculation in my program is like this:

```
// this is blinn phong
vec3 halfDir = normalize(lightDir + viewDir);
float specAngle = max(dot(halfDir, normal), 0.0);
specular = pow(specAngle, Shininess/4.0);
```

  And this part is for calculating Phong reflection:

```
vec3 color = vec3(0.01) + lambertian * diffuseMap + specular * Specular;
```



(output image for this two parts)

- Implement a hierarchy of transformations (a “scene graph”) to place objects relative to others.

This part is about creating a ParentID to implement relative transform. When a node is a root node (ie. It has no parent), the ParentID is set to -1. This forms a tree of transformations. Call AddMeshInstance() to creates an “Instance” of a mesh, which has a MeshID and a TransformID. I create a ParentID in Transform and set it to -1 in AddMeshInstance() to break the loop for the parent transforms of the instance.

```
struct Transform
{
    glm::vec3 Scale;
    glm::vec3 RotationOrigin;
    glm::quat Rotation;
    glm::vec3 Translation;
    int ParentID;
};

void AddMeshInstance(
    Scene* scene,
    uint32_t meshID,
    uint32_t* newInstanceID)
{
    Transform newTransform;
    newTransform.Scale = glm::vec3(1.0f);
    newTransform.ParentID = -1;

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

    Instance newInstance;
    newInstance.MeshID = meshID;
    newInstance.TransformID = newTransformID;

    uint32_t tmpNewInstanceID = scene->Instances.insert(newInstance);
    if (newInstanceID)
    {
        *newInstanceID = tmpNewInstanceID;
    }
}
```

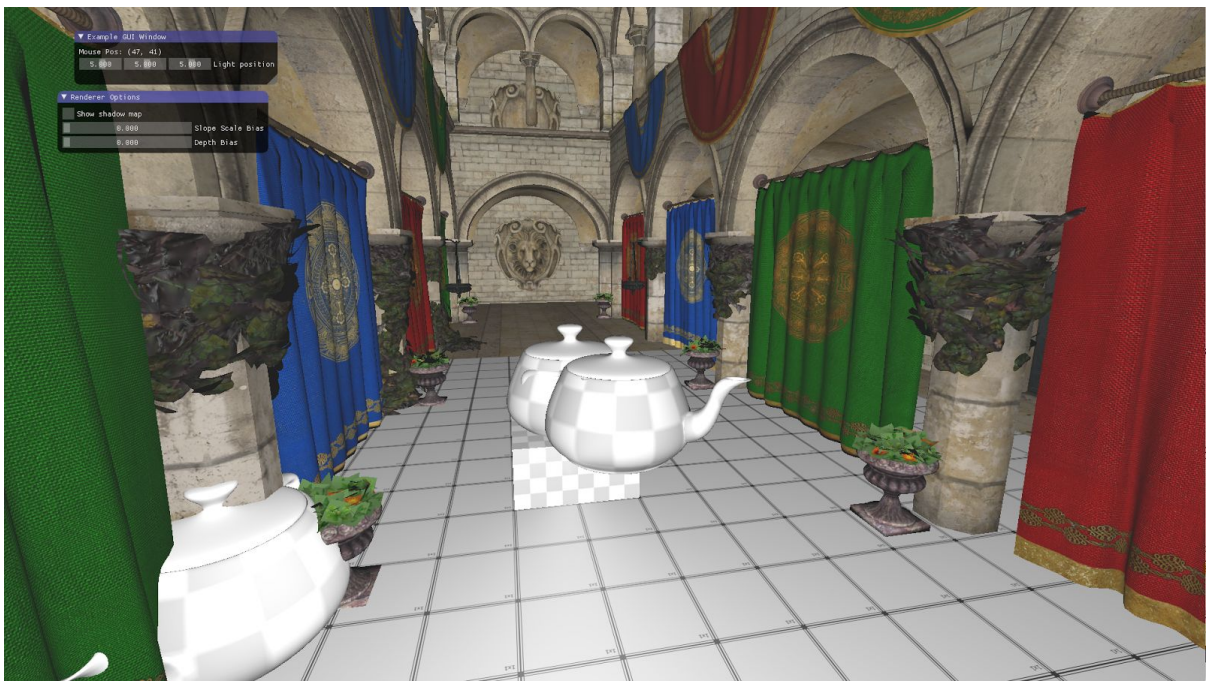
Place one cube and one teapot on the cube. And they both have an unique instanceID. Then place two teapots, one as a parent and another one as a child. Making child teapot rotate around parent teapot:

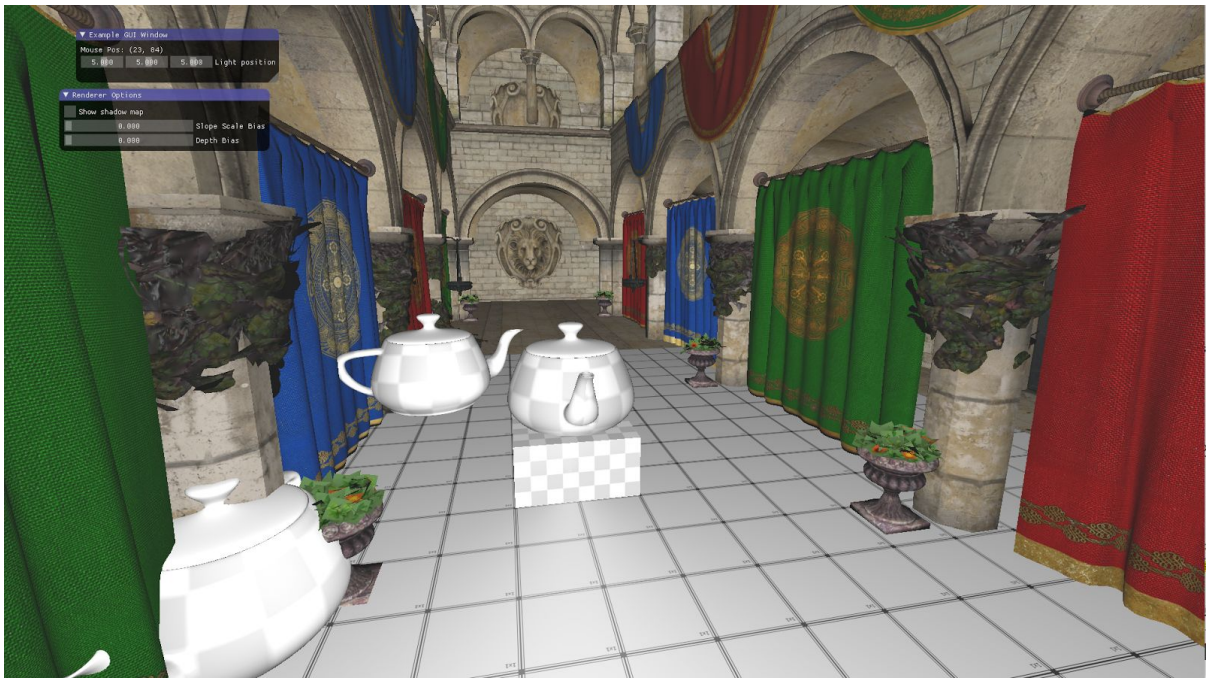
```
{
    uint32_t newInstanceID1;
    AddMeshInstance(mScene, loadedMeshID, &newInstanceID1);
    uint32_t newTransformID1 = scene->Instances[newInstanceID1].TransformID;
    scene->Transforms[newTransformID1].Translation += glm::vec3(3.0f, 1.0f, 4.0f);
    scene->Transforms[newTransformID1].ParentID = -1;


    uint32_t newInstanceID2;
    AddMeshInstance(mScene, loadedMeshID, &newInstanceID2);
    uint32_t newTransformID2 = scene->Instances[newInstanceID2].TransformID;
    mSpinningTransformIDs.push_back(newTransformID2);
    scene->Transforms[newTransformID2].Translation += glm::vec3(3.0f, 1.0f, -4.0f);
    scene->Transforms[newTransformID2].RotationOrigin = -scene->Transforms[newTransformID2].Translation;
    scene->Transforms[newTransformID2].ParentID = newTransformID1;
}
```

Finally update transforms to implement animation. Setting mSpinningTransformID to the ID of a transform for refering to an object during updating.

```
float angularVelocity = 30.0f; // rotating at 30 degrees per second
    for (uint32_t mSpinningTransformID : mSpinningTransformIDs)
        mScene->Transforms[mSpinningTransformID].Rotation *=
            glm::angleAxis(
                glm::radians(angularVelocity * deltaTime),
                glm::vec3(0.0f, 1.0f, 0.0f));
```

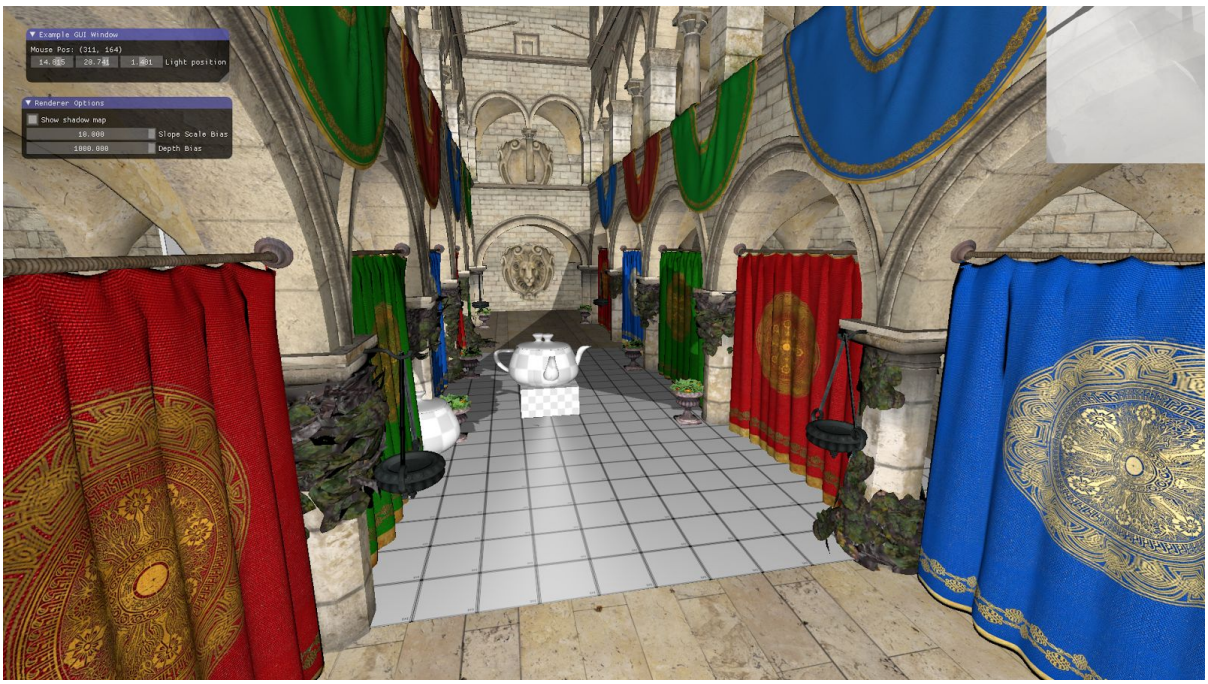


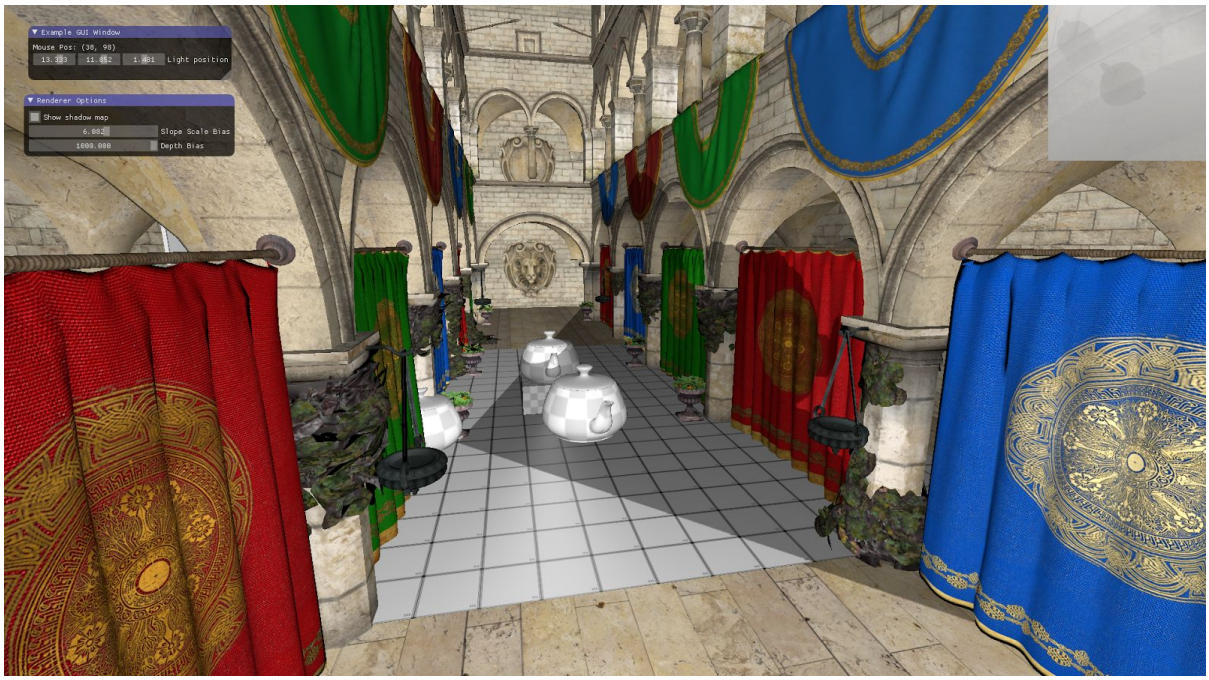


(The teapot is rotating)

- Implement a directional shadow map (from the sun.)

First create a shadow map depth texture and attach it to a framebuffer object. Then loading the scene shader. Setting the first pass for rendering the shadow map's depth texture, which depends on the point of view. Create a shadow.vert to render objects into the depth buffer from the shadow's perspective. Afrerward, setting the second pass for rendering the scene while using the shadow map depth texture as input to test whether points are in shadow or not. Passing the lightOffsetMatrix to the second pass and incorporate the shadowing into lighting computation.





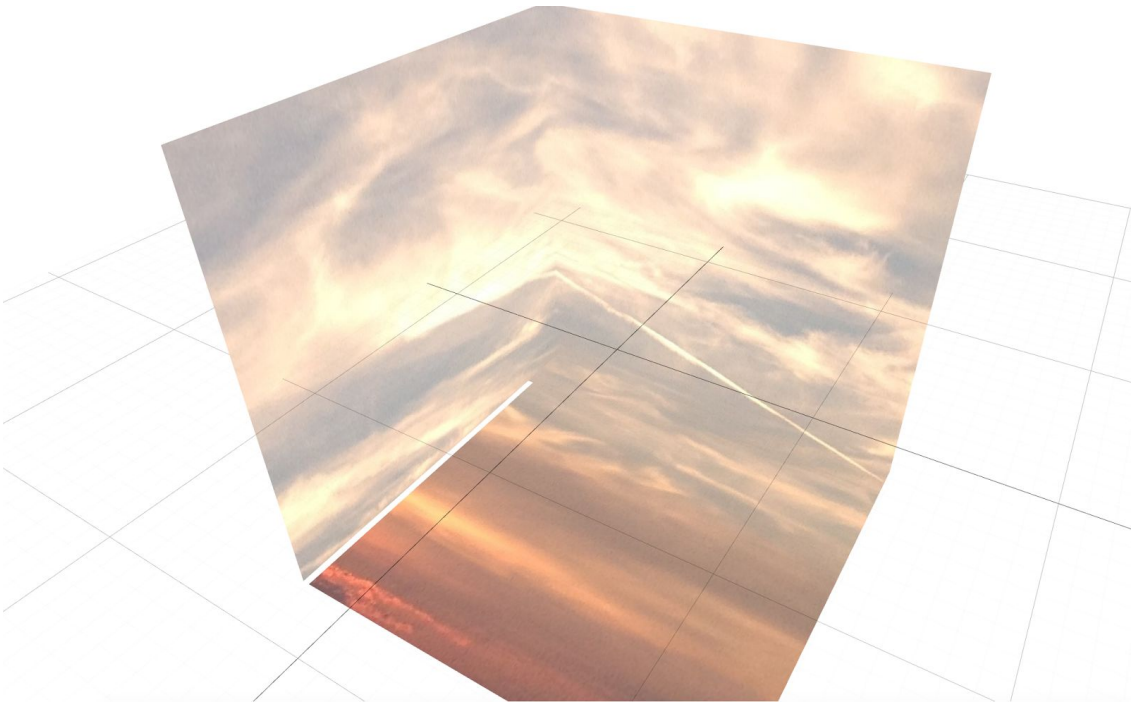
(Adjusting light position and render options to change the shadow)

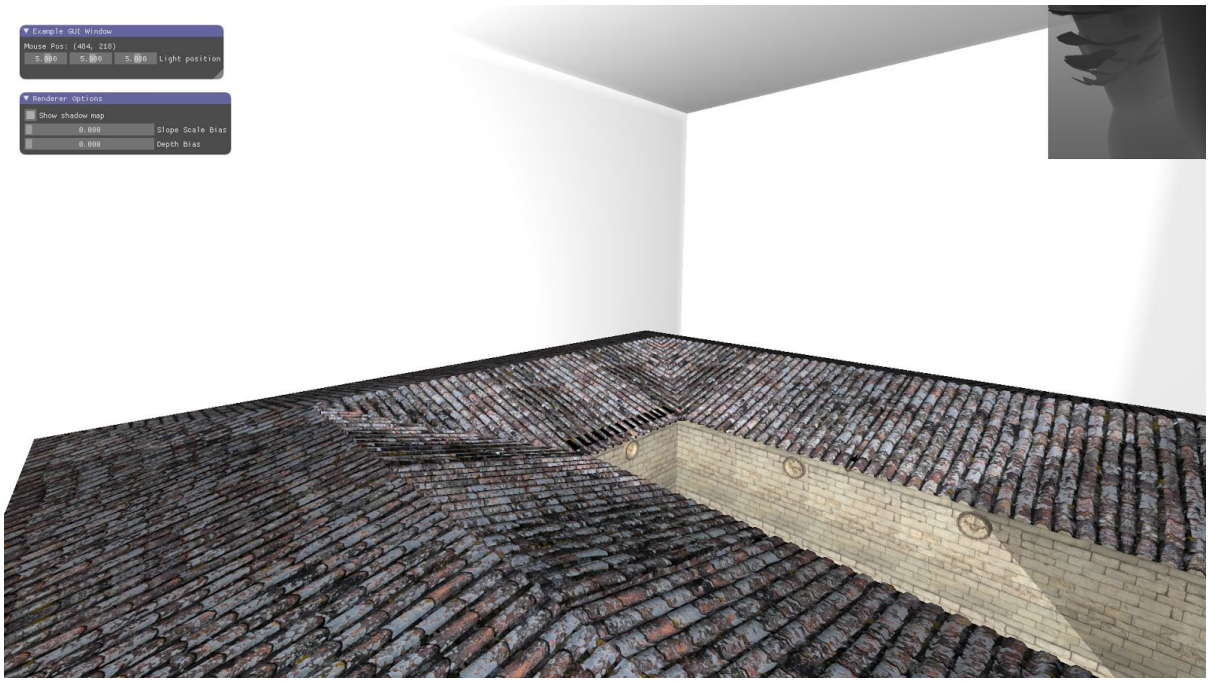
- Render the Sponza scene

  Downloading the sponza_obj.rar and sponza_textures.rar, add them to the assets folder. Changing all “\” to “/” in sponza.mtl. Then adding the Sponza meshes to the scene by loading it in Simulation::Init(). Finally, reset the initial position of camera to see center area of the model.

- Implement a skybox

  I try to create a skybox.obj and skybox.mtl to insert them as the same way as inserting sponza. But the output image dosen’t have any picture.




Example GUI Window
Mouse Pos: (404, 210)
5.000
5.000
5.000
Light position
Renderer Options
Show shadow map
0.000
Slope Scale Bias
0.000
Depth Bias