CSCI3260
Principles of Computer Graphics

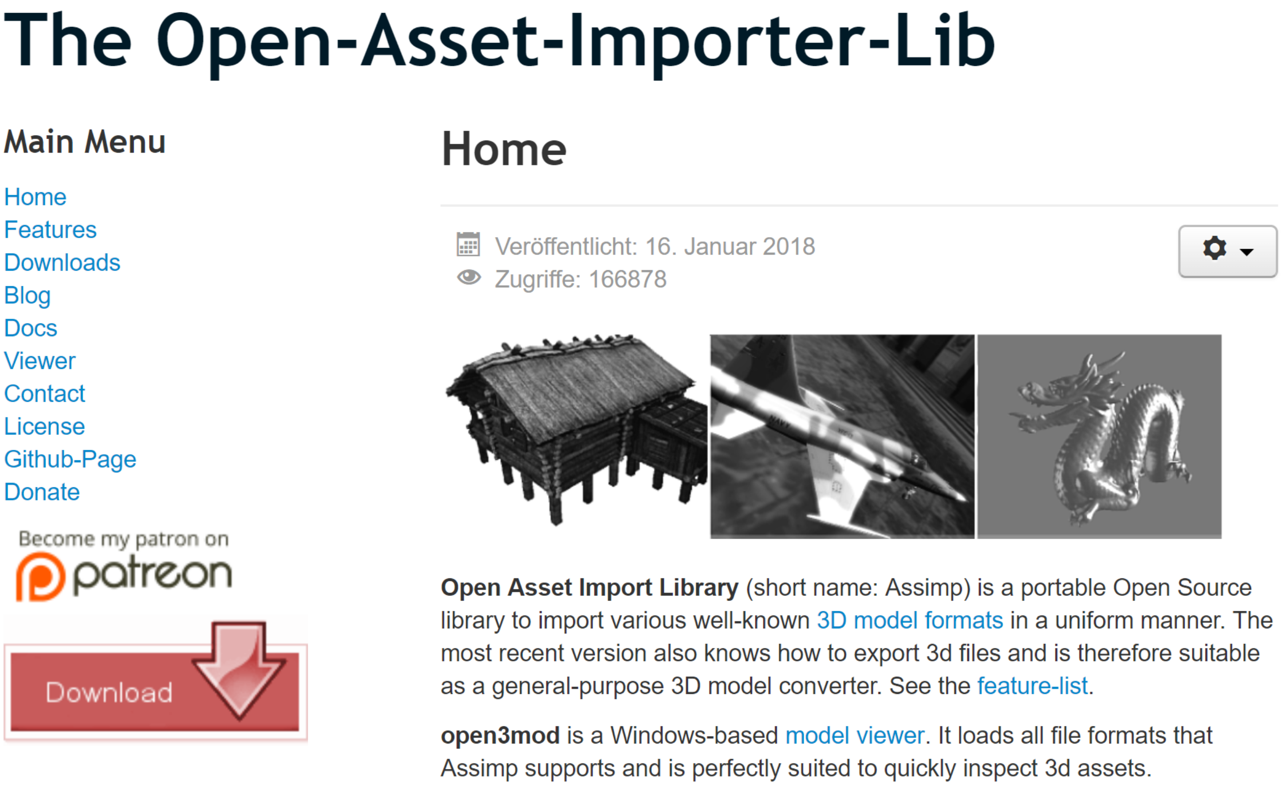
----------Tutorial 8
Meng Ying

OUTLINE

- Load model using **Open Asset Import Library**
- Shadow mapping

Assimp

- A very popular model importing library out there is called Assimp that stands for Open Asset Import Library



The Open-Asset-Importer-Lib

Home

Veröffentlicht: 16. Januar 2018
Zugriffe: 166878

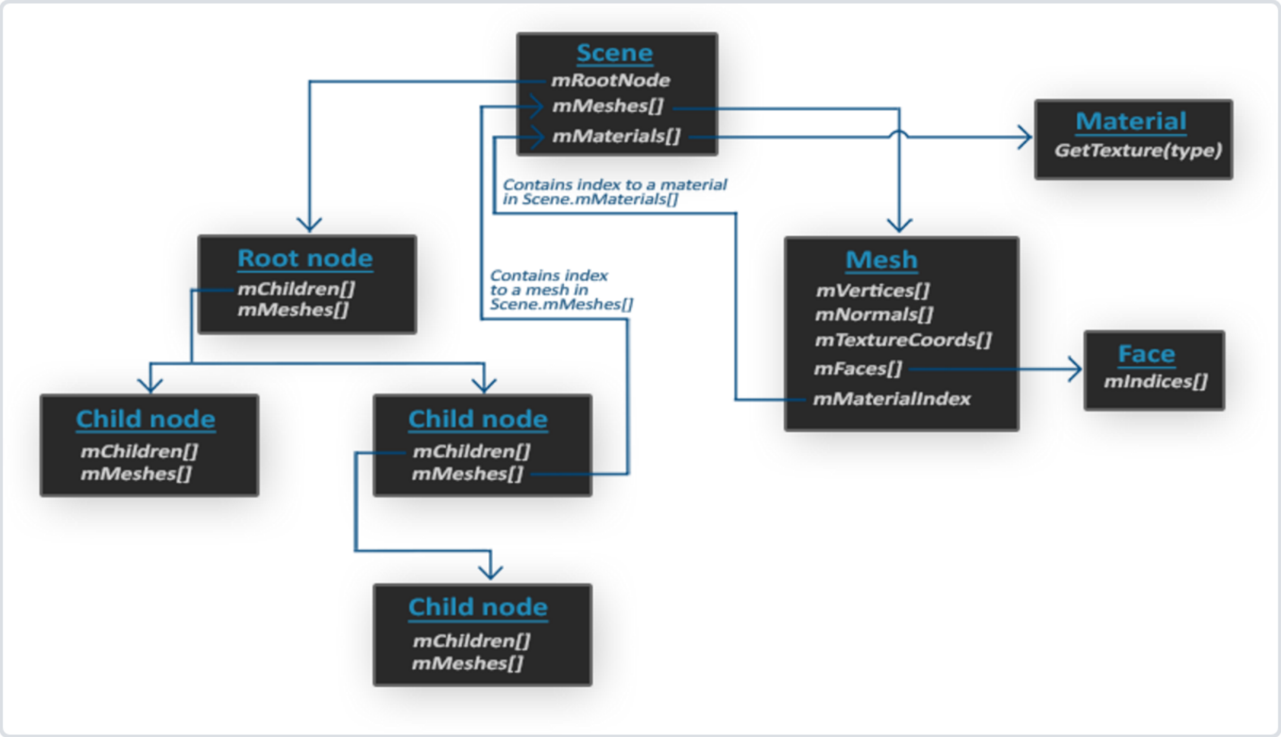
Open Asset Import Library (short name: Assimp) is a portable Open Source library to import various well-known 3D model formats in a uniform manner. The most recent version also knows how to export 3d files and is therefore suitable as a general-purpose 3D model converter. See the feature-list.

open3mod is a Windows-based model viewer. It loads all file formats that Assimp supports and is perfectly suited to quickly inspect 3d assets.

http://assimp.org/index.php/downloads

Assimp

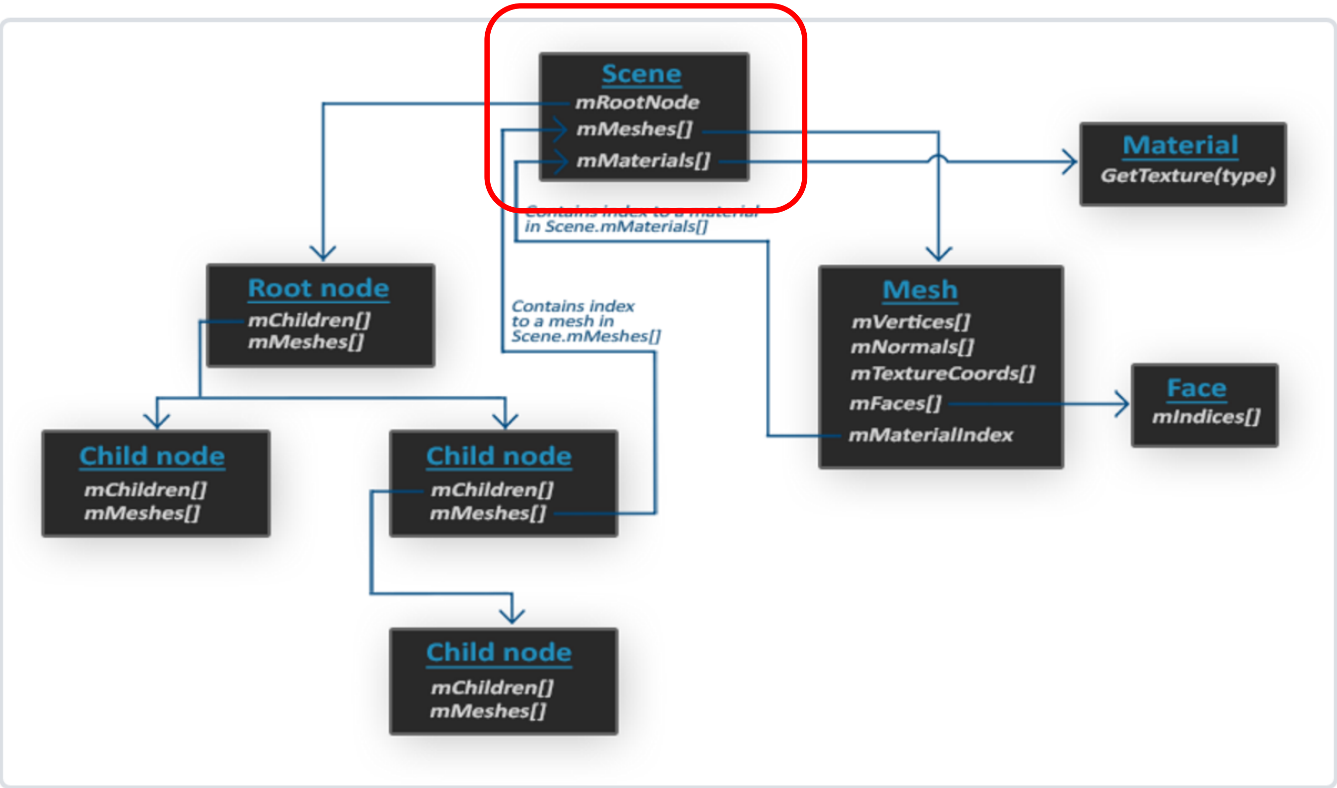
- Assimp is able to import dozens of different model file formats by loading all the model's data into Assimp's **generalized** data structures.(eg. obj, ply, stl)
- Retrieve all the data we need from Assimp's data structures.



Assimp's data structures

Assimp

All the data of the scene/model is contained in the Scene object like all the materials and the meshes.



Scene
mRootNode
mMeshes[]
mMaterials[]

Material
GetTexture(type)

Contains index to a material in Scene.mMaterials[]

Root node
mChildren[]
mMeshes[]

Contains index to a mesh in Scene.mMeshes[]

Mesh
mVertices[]
mNormals[]
mTextureCoords[]
mFaces[]
mMaterialIndex

Face
mIndices[]

Child node
mChildren[]
mMeshes[]

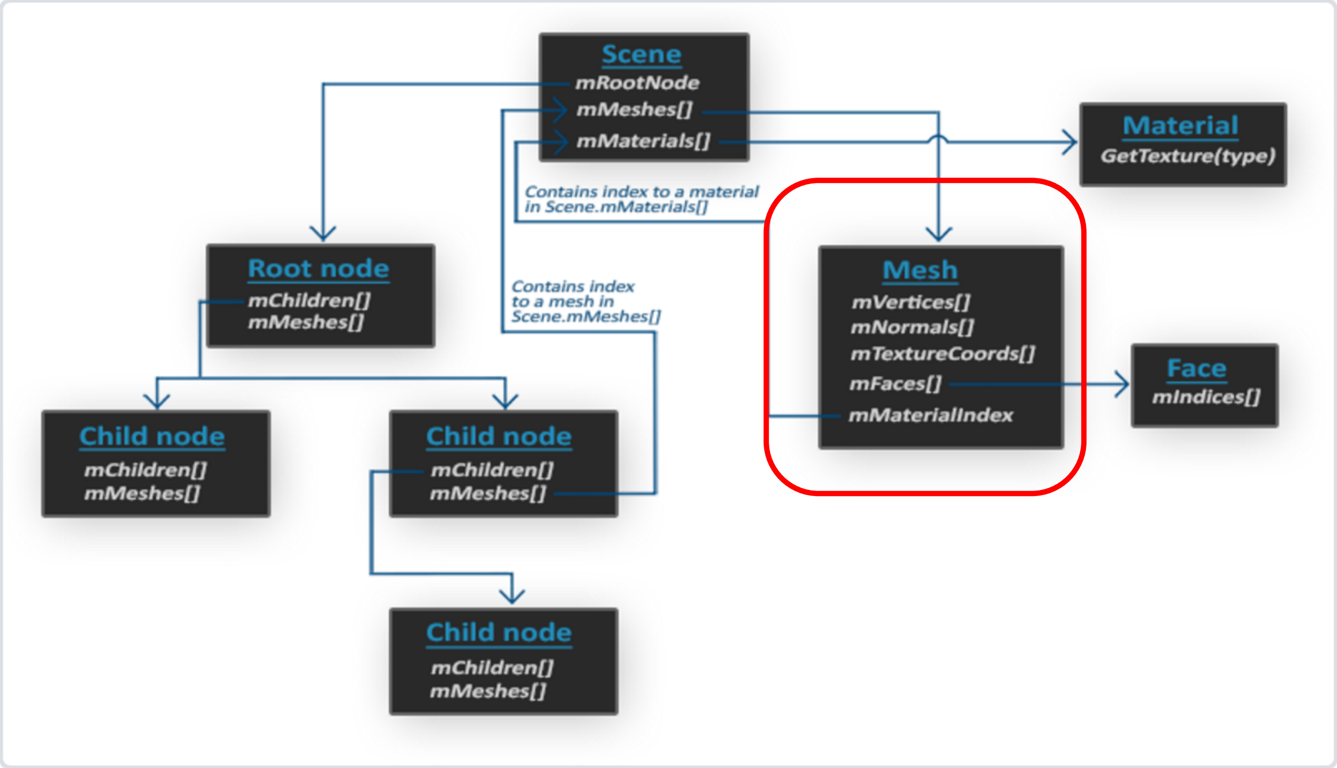
Child node
mChildren[]
mMeshes[]

Child node
mChildren[]
mMeshes[]

Assimp's data structures

Assimp

A Mesh object itself contains all the relevant data required for rendering, think of vertex positions, normal vectors, texture coordinates, faces and the material of the object.



Assimp's data structures

Assimp

- Process when we use Assimp

- Load an object into a Scene object
- Recursively retrieve the corresponding Mesh objects from each of the nodes
- Process each Mesh object to retrieve the vertex data, indices and its material properties.
- The result is then a collection of mesh data that we want to contain in a single Model object.

Assimp

- Details

- Build Assimp

  https://learnopengl.com/Model-Loading/Assimp

- Mesh introduction

  http://learnopengl.com/Model-Loading/Mesh

- Load model

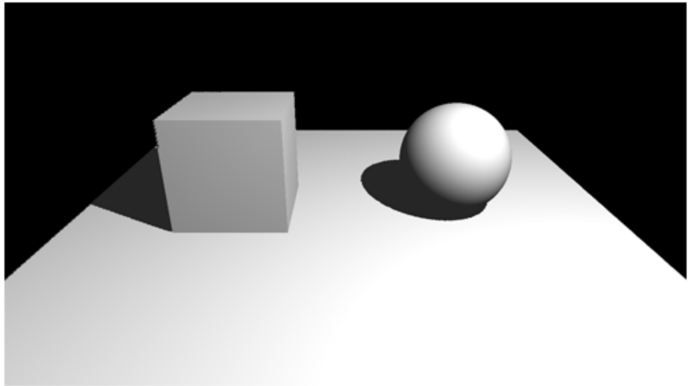
  http://learnopengl.com/Model-Loading/Model

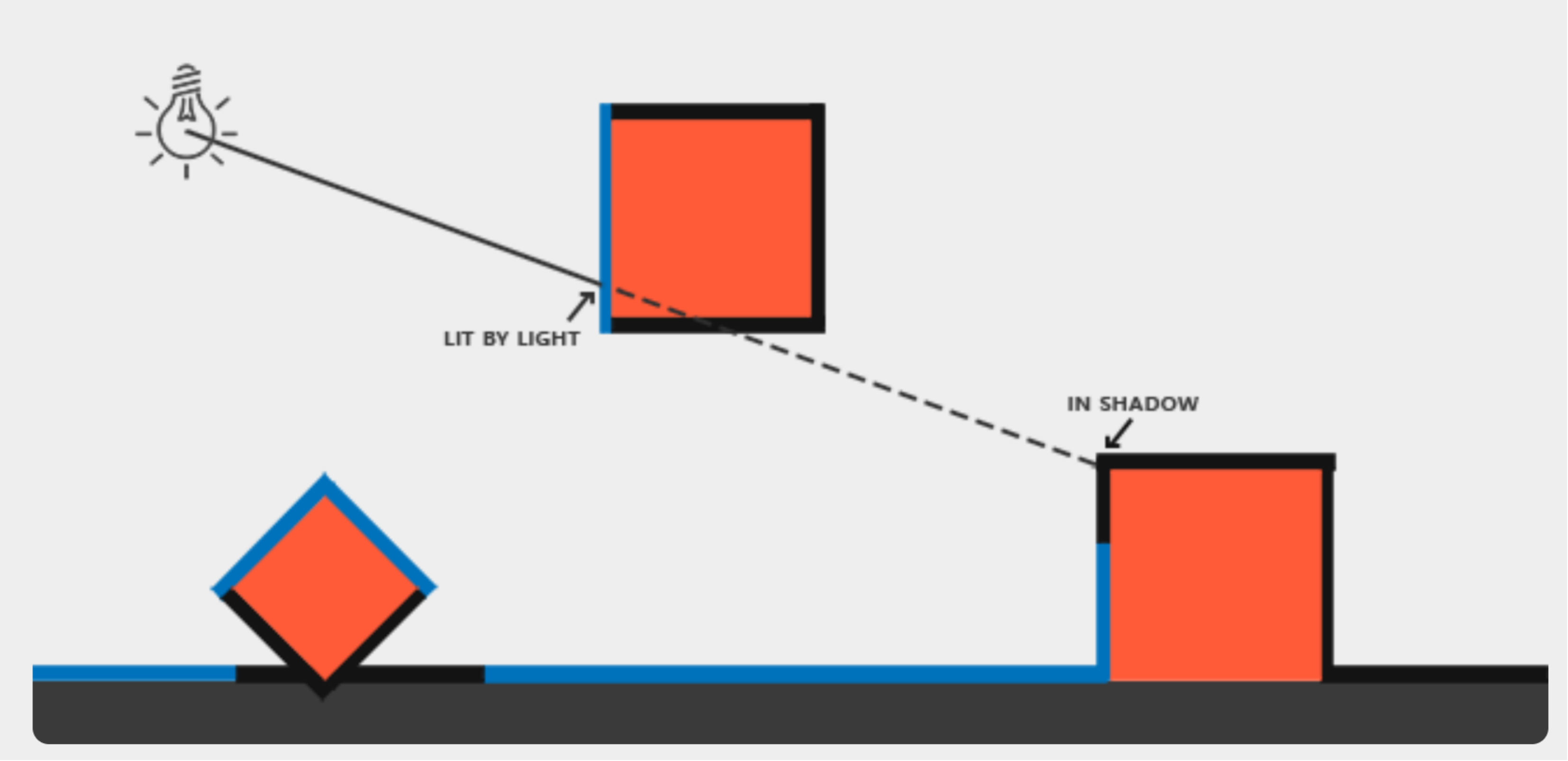
OUTLINE

- Load model using Open Asset Import Library
- Shadow mapping

Shadow

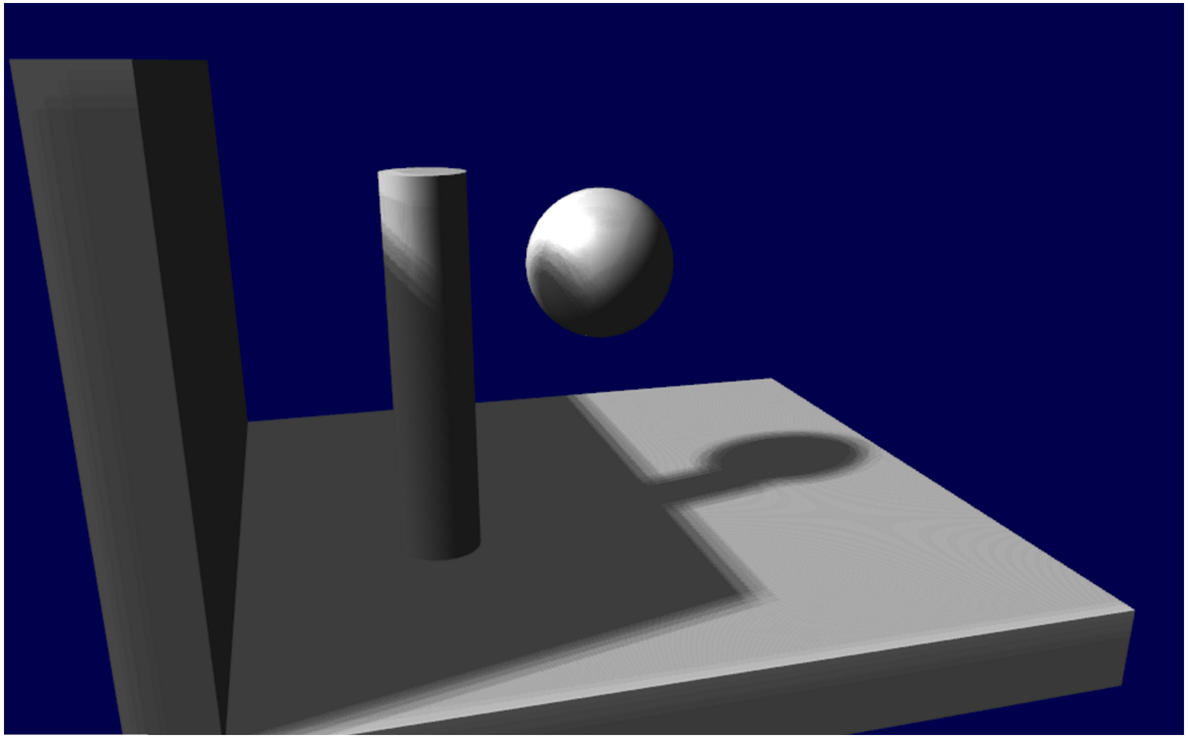
- What is shadow
  - Light cannot pass





Shadow mapping

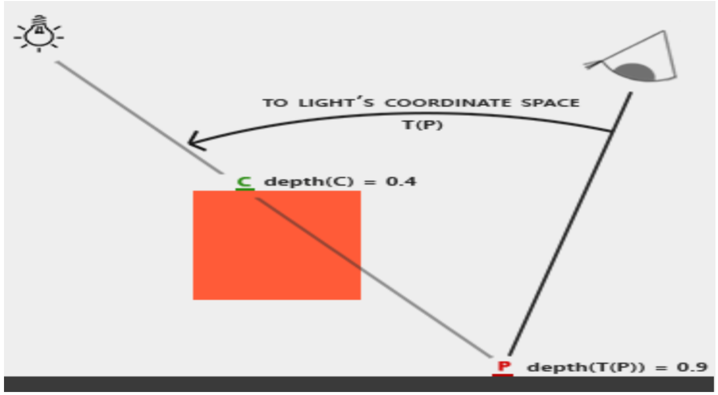
- Shadow mapping
  - Dynamic shadows
  - Demo video: https://www.youtube.com/watch?v=XF30cLr_-V8



Shadow mapping

➢ Algorithm [two passes]

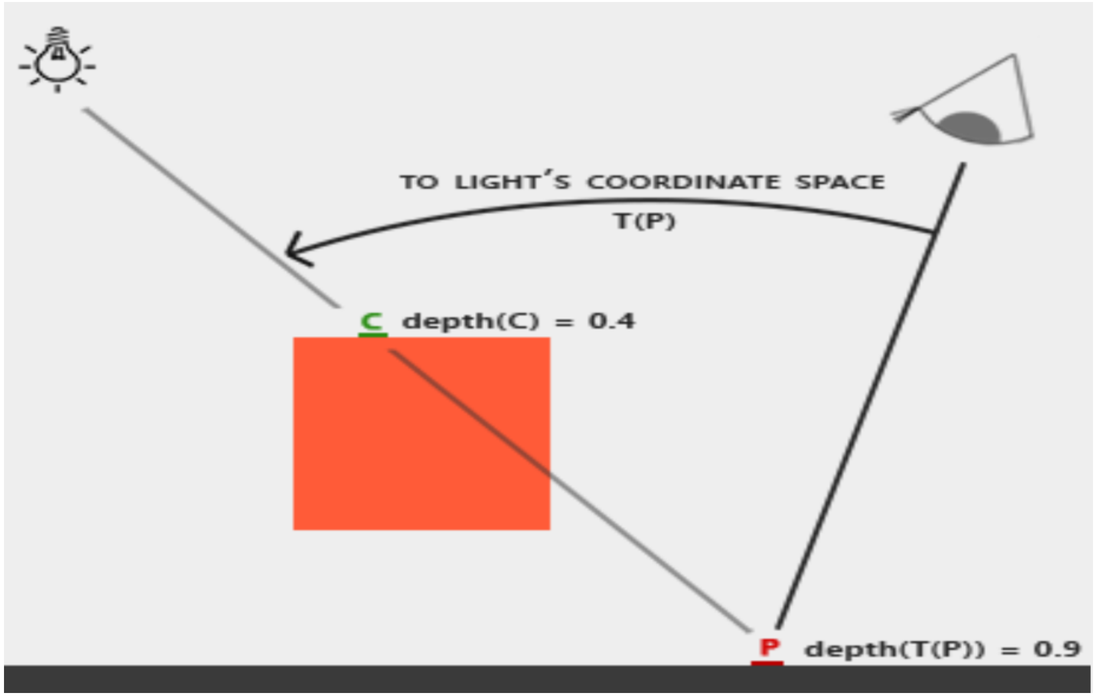
1. Rendering the shadow map
   - The scene is rendered from the point of view of the light
   - Only the depth of each viewed fragment is computed



2. Usually rendering the scene using shadow map
   - The scene is rendered as usual, but with an extra test to see if the current fragment is in the shadow.

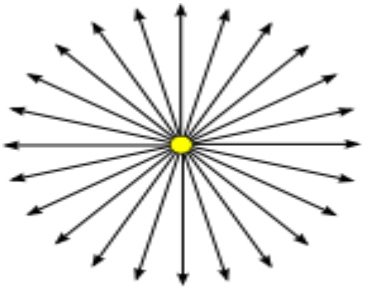
Shadow mapping

- extra test
  - If the current sample is further from the light than the shadow map at the same point, this means that the scene contains an object that is closer to the light.
  - In other words, the current fragment is in the shadow.

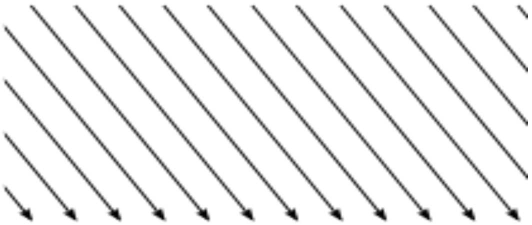


Rendering the shadow map

- Light Source
  - Only consider directional lights - lights that are so far away that all the light rays can be considered parallel.
  - Rendering the shadow map is done with an orthographic projection matrix.
  - An orthographic matrix is just like a usual perspective projection matrix, except that no perspective is considered - an object will look the same whether it's far or near the camera.

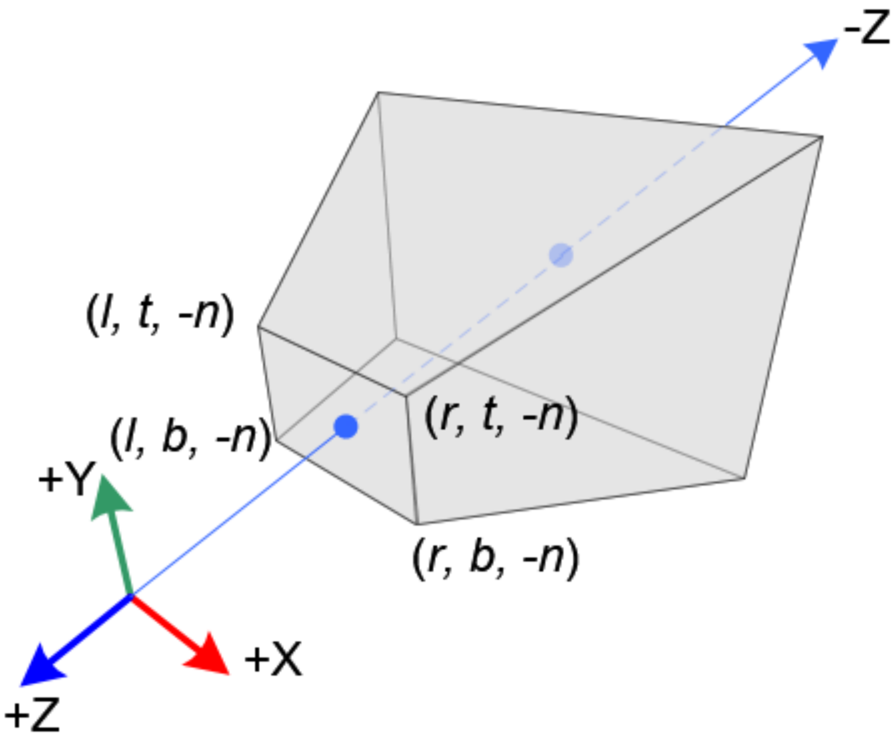


POINT LIGHT
emits light in
all directions.

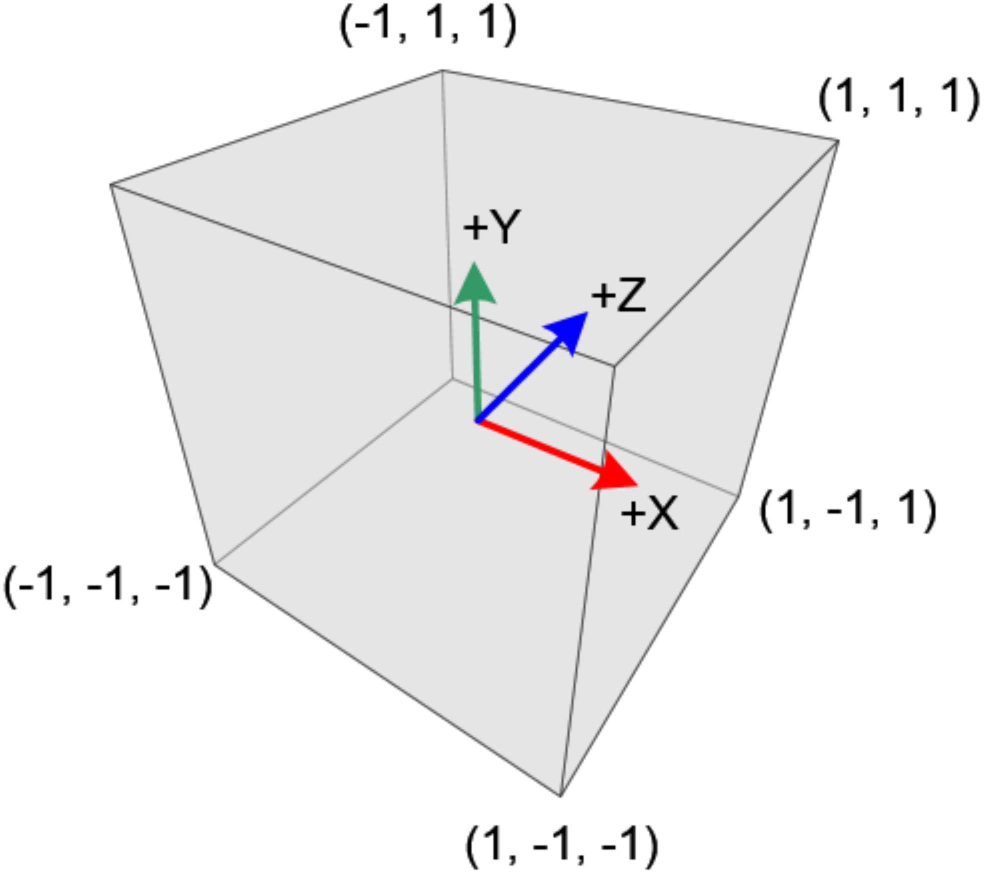


DIRECTIONAL LIGHT
has parallel light rays, all
from the same direction.

Rendering the shadow map



perspective projection



orthographic projection

Rendering the shadow map

- Setting up the render target - sendDataToOpenGL()

- Use framebuffer

```
GLuint shadowFrameBuffer = 0;
glGenFramebuffers(1, & shadowFrameBuffer);
```

- Create depth texture

```
 GLuint depthTexture;
 glGenTextures(1, &depthTexture);
 glBindTexture(GL_TEXTURE_2D, depthTexture);
 glTexImage2D(GL_TEXTURE_2D, 0,GL_DEPTH_COMPONENT, 1024, 1024,
0,GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

Rendering the shadow map

➢ Setting up the render target - sendDataToOpenGL()

- Set "renderedTexture" as our colour attachement #0

```
glBindFramebuffer(GL_FRAMEBUFFER, shadowFrameBuffer);
glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, depthTexture, 0);
glDrawBuffer(GL_NONE); // No color buffer is drawn to.
```

- Always check that our framebuffer is OK

```
if(glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
{cout << "FrameBuffer does not complete!" << endl;}
```

Rendering the shadow map

➢ Install new shadow map shader

- Create the function which is similar to installShaders()

```
void installShadowMapShaders(){
…
string temp = readShaderCode("ShadowVertexShader.glsl");
//new vertex shader file, should also create new shader file for fragment.
adapter[0] = temp.c_str();
glShaderSource(vertexShaderID, 1, adapter, 0);
…
shadowMapProgramID = glCreateProgram(); //new program ID
glAttachShader(shadowMapProgramID, vertexShaderID);
glAttachShader(shadowMapProgramID, fragmentShaderID);
glLinkProgram(shadowMapProgramID);
if (!checkProgramStatus(shadowMapProgramID))
        return;
…
```

- Just like using multiple shaders, details in the next tutorial

Rendering the shadow map

- Render shadow map - paintGL()

- Using shadow map shader

```
glUseProgram(shadowMapProgramID);
glBindFramebuffer(GL_FRAMEBUFFER, shadowFrameBuffer);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glViewport(0, 0, 1024, 1024);
drawScene();
```

Rendering the shadow map

- Render shadow map - paintGL()

- Create the depth MVP matrix which is used to render the scene from the light's point of view

```
glm::vec3 lightInvDir = glm::vec3(0.5f,2,2);

glm::mat4 depthProjectionMatrix =
glm::ortho<float>(-10,10,-10,10,-10,20);



glm::mat4 depthViewMatrix =
glm::lookAt(lightInvDir, glm::vec3(0,0,0),
glm::vec3(0,1,0));

glm::mat4 depthModelMatrix = glm::mat4(1.0);
```

Light's point

The Projection matrix is an orthographic matrix which will encompass everything in the axis-aligned box (-10,10), (-10,10),(-10,20) on the X,Y and Z axes. These values are made so that our entire scene is always visible

Look at origin

The Model matrix (transformation matrix): depends on different models

Rendering the shadow map

➢ Render shadow map - paintGL()

- Create the depth MVP matrix

```
glm::mat4 depthMVP = depthProjectionMatrix * depthViewMatrix * depthModelMatrix;
```

- Send different transformations to different models to the currently bound shader, (shadow map shader) and render shadow maps

```
GLuint depthMatrixLocation = glGetUniformLocation(shadowMapProgramID,
"depthMVP ");
glBindVertexArray(dolphinObj); //bind model information
glUniformMatrix4fv(depthMatrixLocation, 1, GL_FALSE, &depthMVP[0][0])
//send its transformation to currently bound shader
glDrawElements(GL_TRIANGLES, dolphinObj.indices.size(),
GL_UNSIGNED_INT, 0);//render the shadow map
```

Rendering the shadow map

- ➢ The new shadow map vertexshader
- Compute the vertex’ position in homogeneous coordinates

```
// Input vertex data, different for all executions of this shader.
in layout(location = 0) vec3 position_modelSpace;

// Values that stay constant for the whole mesh.
uniform mat4 depthMVP;

void main(){
 gl_Position =  depthMVP * vec4(position_modelSpace,1);
}
```

Rendering the shadow map

- The new shadow map fragmentshader

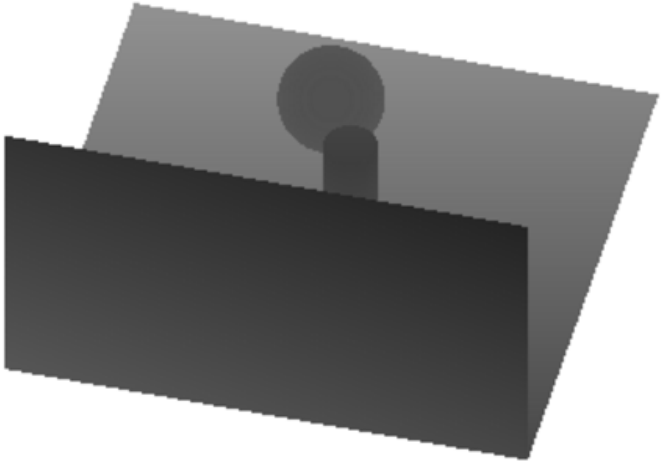
- Writes the depth of the fragment at location 0 (i.e. in our depth texture)

```
// Ouput data
out layout(location = 0) float fragmentdepth;

void main(){
    // Not really needed, OpenGL does it anyway
    fragmentdepth = gl_FragCoord.z;
}
```

Rendering the shadow map

- Result



A dark color means a small z ; hence, the upper-right corner of the wall is near the camera. At the opposite, white means z=1 (in homogeneous coordinates), so this is very far.

Using the shadow map

Basic shader

- For each fragment that we compute, we must test whether it is “behind” the shadow map or not.
- To do this, we need to compute the current fragment’s position in the same space that the one we used when creating the shadow map.
- So we need to transform it once with the usual MVP matrix, and another time with the depthMVP matrix.

```
//using the basic shader
glUseProgram(programID);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glViewport(0, 0, 512, 512);
//set the usual MVP matrix and lighting information
…
//set the depth MVP matrix
…
```

Using the shadow map

➢ Basic shader - A little trick about depth MVP

- Multiplying the vertex' position by depthMVP will give homogeneous coordinates, which are in [-1,1] ;
- But texture sampling must be done in [0,1].
- Example: A fragment in the middle of the screen will be in (0,0) in homogeneous coordinates ; but since it will have to sample the middle of the texture, the UVs will have to be (0.5, 0.5).

➢ Solve

- multiply the homogeneous coordinates by the following matrix

```
glm::mat4 biasMatrix(
0.5, 0.0, 0.0, 0.0,
0.0, 0.5, 0.0, 0.0,
0.0, 0.0, 0.5, 0.0,
0.5, 0.5, 0.5, 1.0
);
glm::mat4 depthBiasMVP = biasMatrix*depthMVP;
```

Divides coordinates by 2 (the diagonal : [-1,1] -> [-0.5, 0.5])
Translates them (the lower row : [-0.5, 0.5] -> [0,1])

Using the shadow map

- ➢ Basic shader - render models
- • Use ‘dolphin’ model as an example

```
//set texture location in shader: obj texture and depth texture used in shadow map
GLuint TextureID = glGetUniformLocation(programID, "objTexture");
GLuint shadowMapID = glGetUniformLocation(programID, "shadowMap");
```

```
glBindVertexArray(dolphinObj); //bind model
information

//bind obj and depth textures
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D,
dolphin_texture);
glUniform1i(TextureID, 0);

glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, depthTexture);
glUniform1i(shadowMapID, 1);
```

```
//Already loaded in sendDataToOpenGL()
dolphin_texture = loadTexture(“dolphin_01.jpg");
```

```
//Already calculated by sendDataToOpenGL() and
shadow FragmentShader
glFramebufferTexture(GL_FRAMEBUFFER,
GL_DEPTH_ATTACHMENT, depthTexture, 0);
```

Using the shadow map

- Basic shader - render models
- Use ‘dolphin’ model as an example

```
//send the usual MVP and depth MVP to shader
…
//draw model
glDrawElements(…);
```

Using the shadow map

- The vertex shader
- output 2 positions
- gl_Position is the position of the vertex as seen from the current camera
- ShadowCoord is the position of the vertex as seen from the light source

```
//get MVP and depthMVP from opengl
…
// Output position of the vertex, in clip space : MVP * position
gl_Position =  MVP * vec4(vertexPosition_modelspace,1);

// Same, but with the light's view matrix
ShadowCoord = DepthBiasMVP * vec4(vertexPosition_modelspace, 1);
```

Using the shadow map

- The fragment shader

```
//get ShadowCoord from vertexshader
  in vec4 shadow_vertexPosition;

//get depth texture (calculated shadow map) from opengl
 uniform sampler2D shadowMap;
```

Using the shadow map

- The fragment shader
- If the current fragment is further than the nearest occluder, this means we are in the shadow (of said nearest occluder)
- texture(shadowMap, ShadowCoord.xy).z is the distance between the light and the nearest occluder
- ShadowCoord.z is the distance between the light and the current fragment

```
float visibility = 1.0;
if ( texture( shadowMap, ShadowCoord.xy ).z  <  ShadowCoord.z){
    visibility = 0.5; //hyperparameter depends on your models
}
```

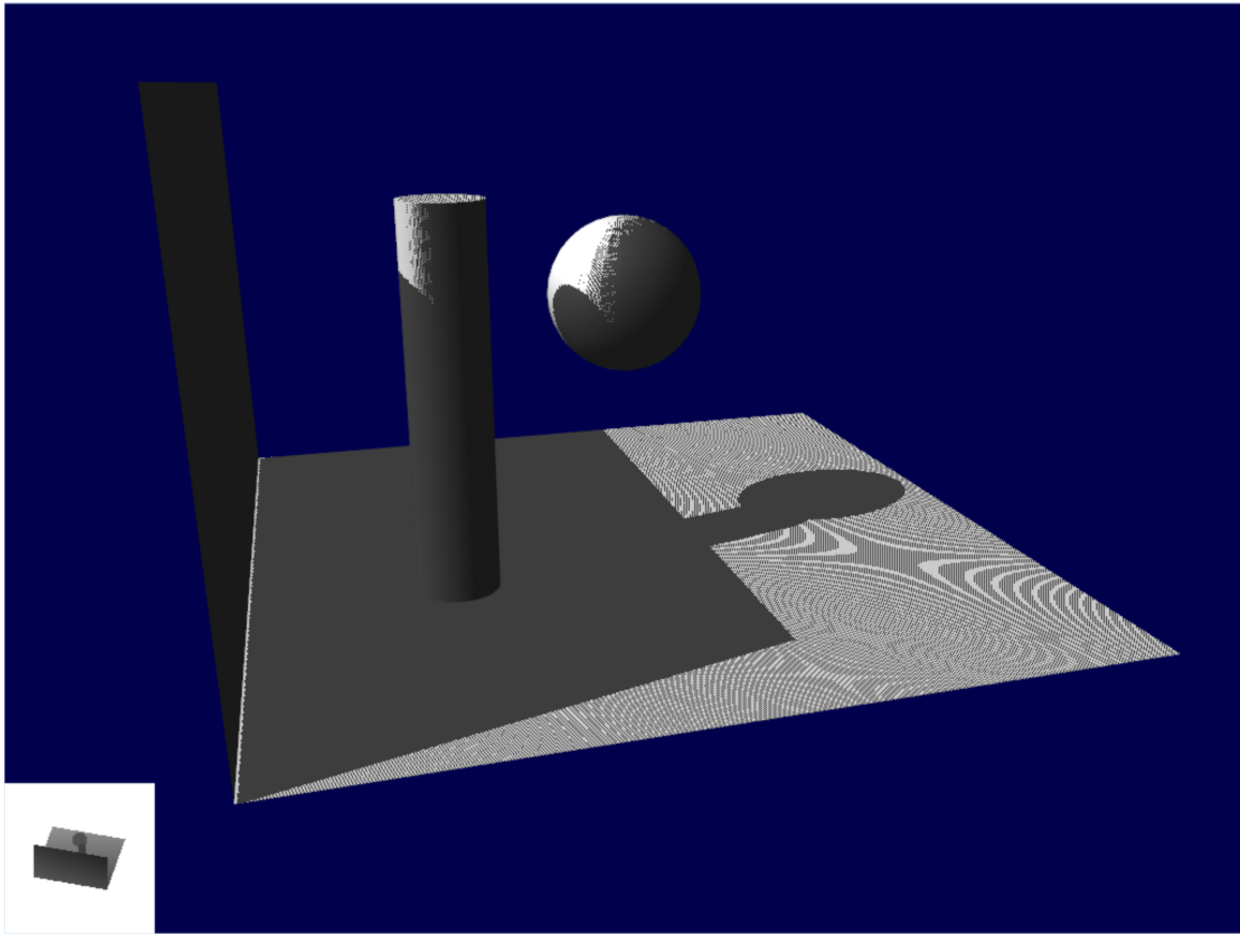
Using the shadow map

- The fragment shader
- Modify shadering

```
color =
 // Ambient : shadow mapping has no effect on ambient light
 MaterialAmbientColor  * AmbientLightColor +
 // Diffuse and specular: use ‘visibility’ to control
// Can set different visibilities to diffuse and specular light
 visibility * MaterialDiffuseColor  * DiffuseLightColor * DiffuseBrightness +
 visibility * MaterialSpecularColor * SpecularLightColor * pow(SpecularBrightness,50);
```

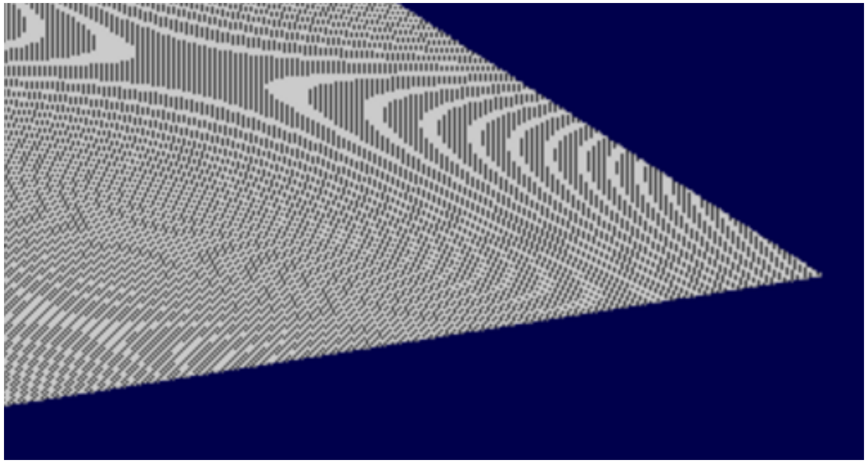
Using the shadow map

- Result
- The global idea it there, but the quality is unacceptable.



Shadow acne

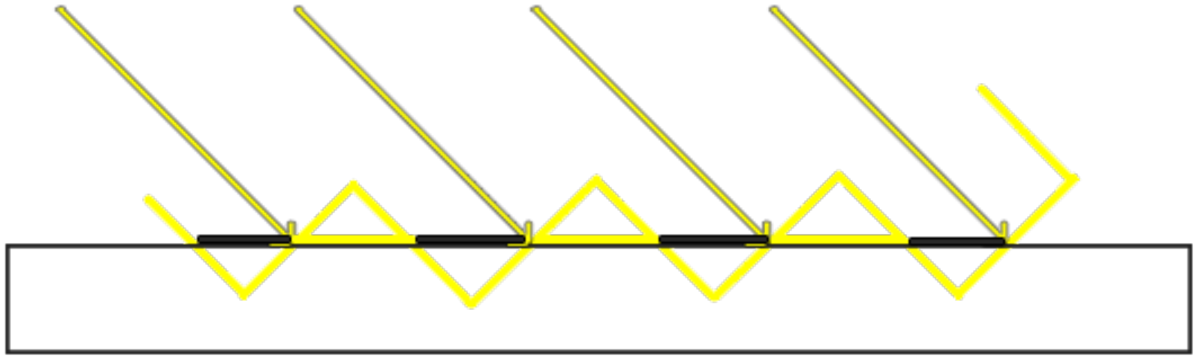
- The most obvious problem is called shadow acne



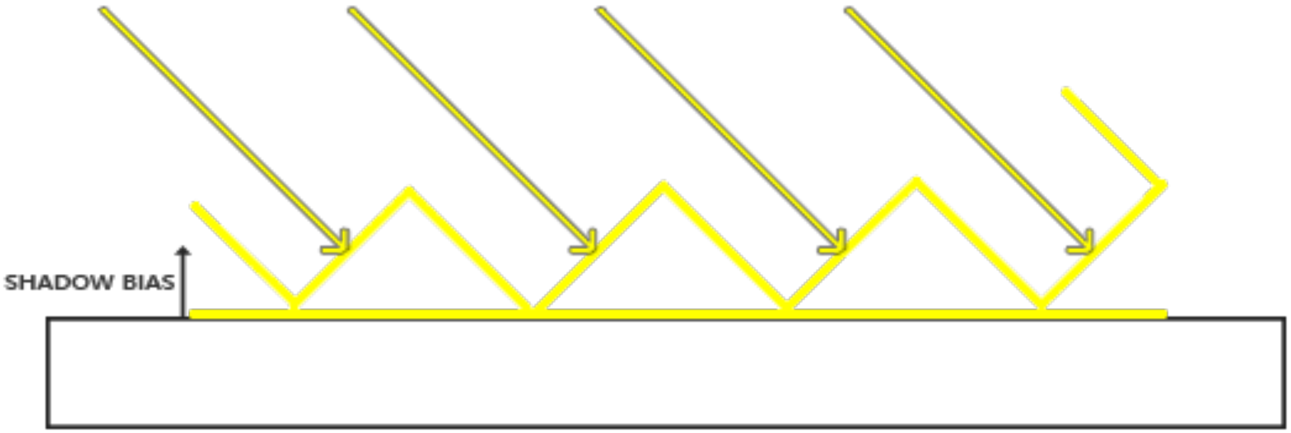
- Solve: add an error margin (a bias)
- we only shade if the current fragment’s depth (again, in light space) is really far away from the lightmap value.

```
float visibility = 1.0;
float bias = 0.005; //hyperparameter depends on your models
if ( texture( shadowMap, ShadowCoord.xy ).z  <  ShadowCoord.z-bias){
    visibility = 0.5;
}
```

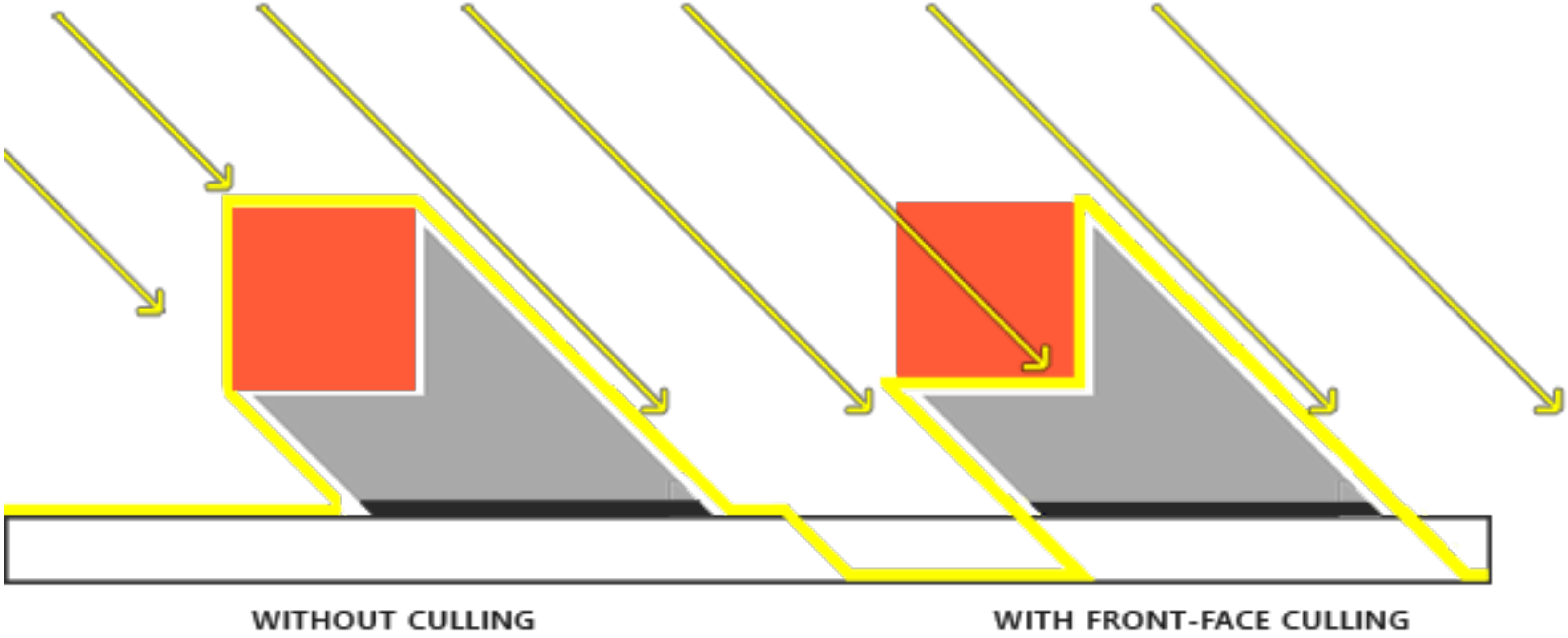
Using the shadow map



Irregular surface



Peter Panning when rendering shadow map

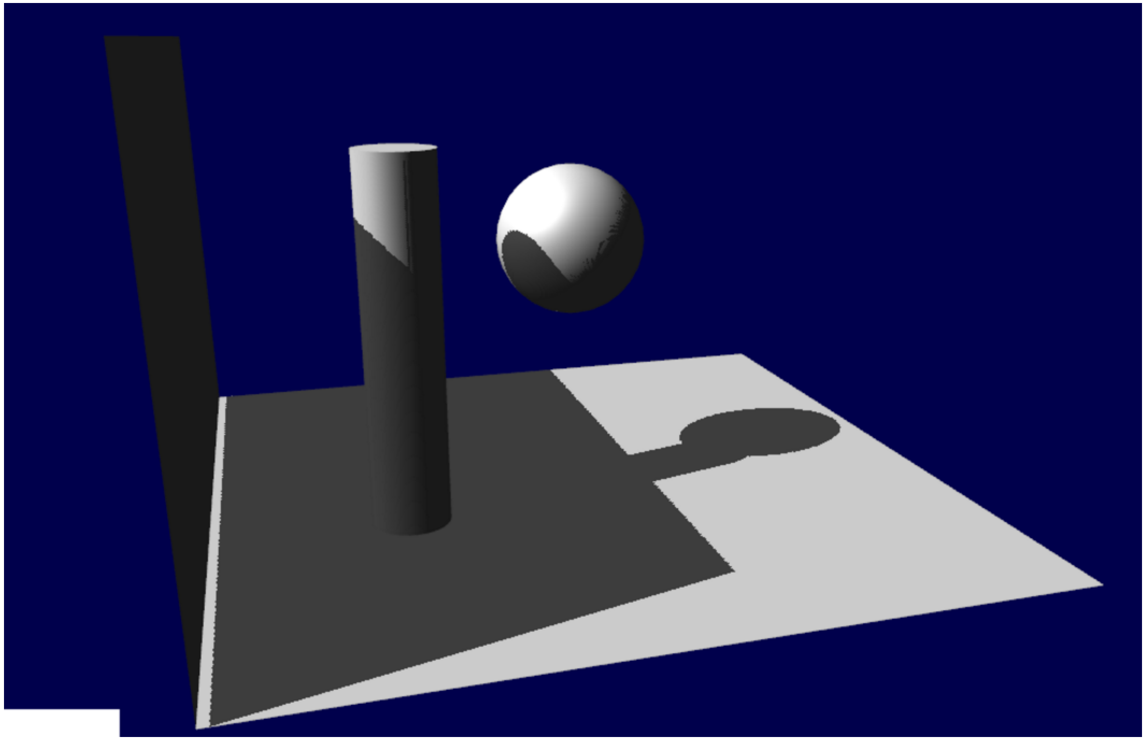


Acne is in the shadow anyway

```
glCullFace(GL_FRONT);
RenderSceneToDepthMap();
glCullFace(GL_BACK); // don't forget to reset original culling face
```

Shadow acne

- Some artifacts remain on the cylinder and on the sphere.

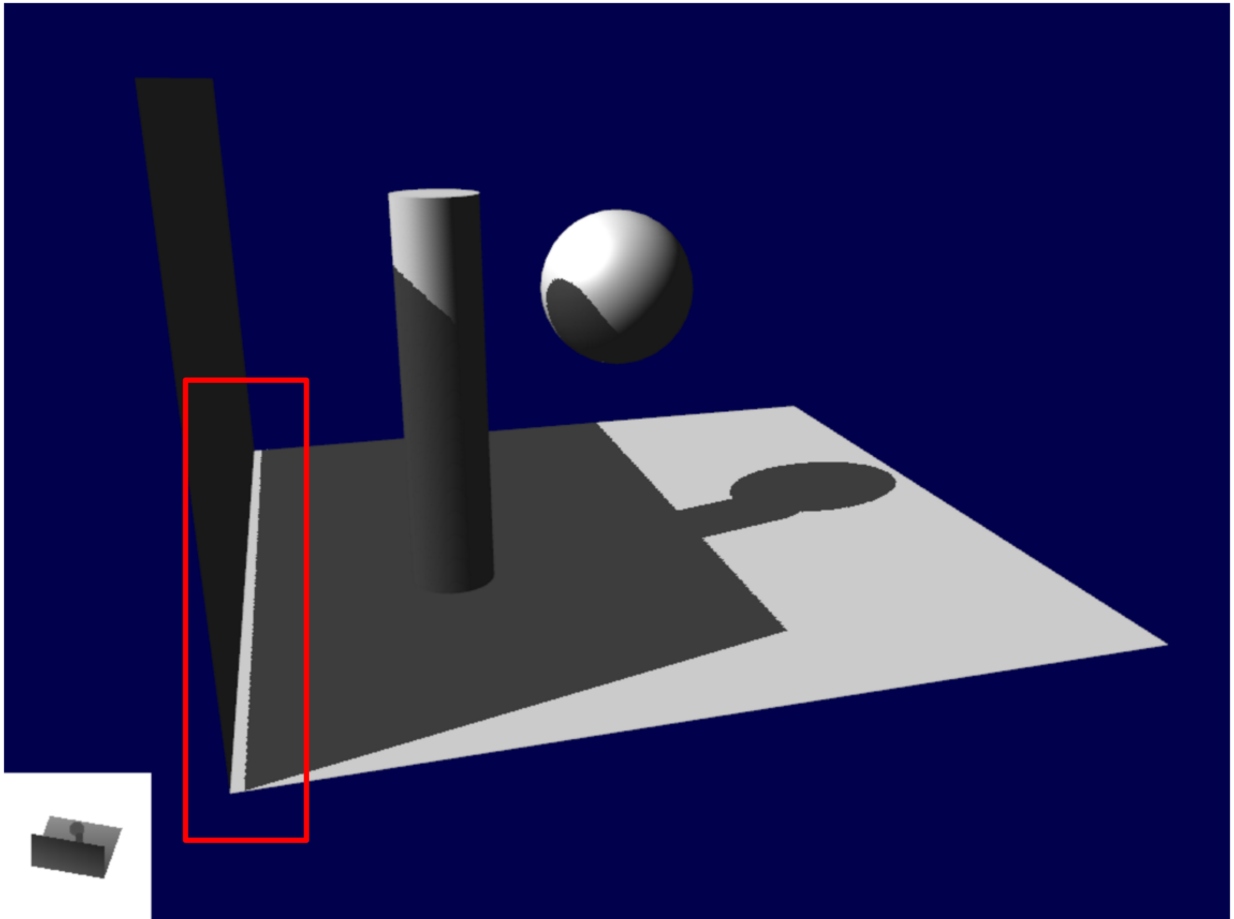


- Solve: modify the bias according to the slope

```
float bias = 0.005*tan(acos(cosTheta));
// cosTheta is dot( normal, lightvector), clamped between 0 and 1
bias = clamp(bias, 0,0.01); //hyperparameters depend on your models
```

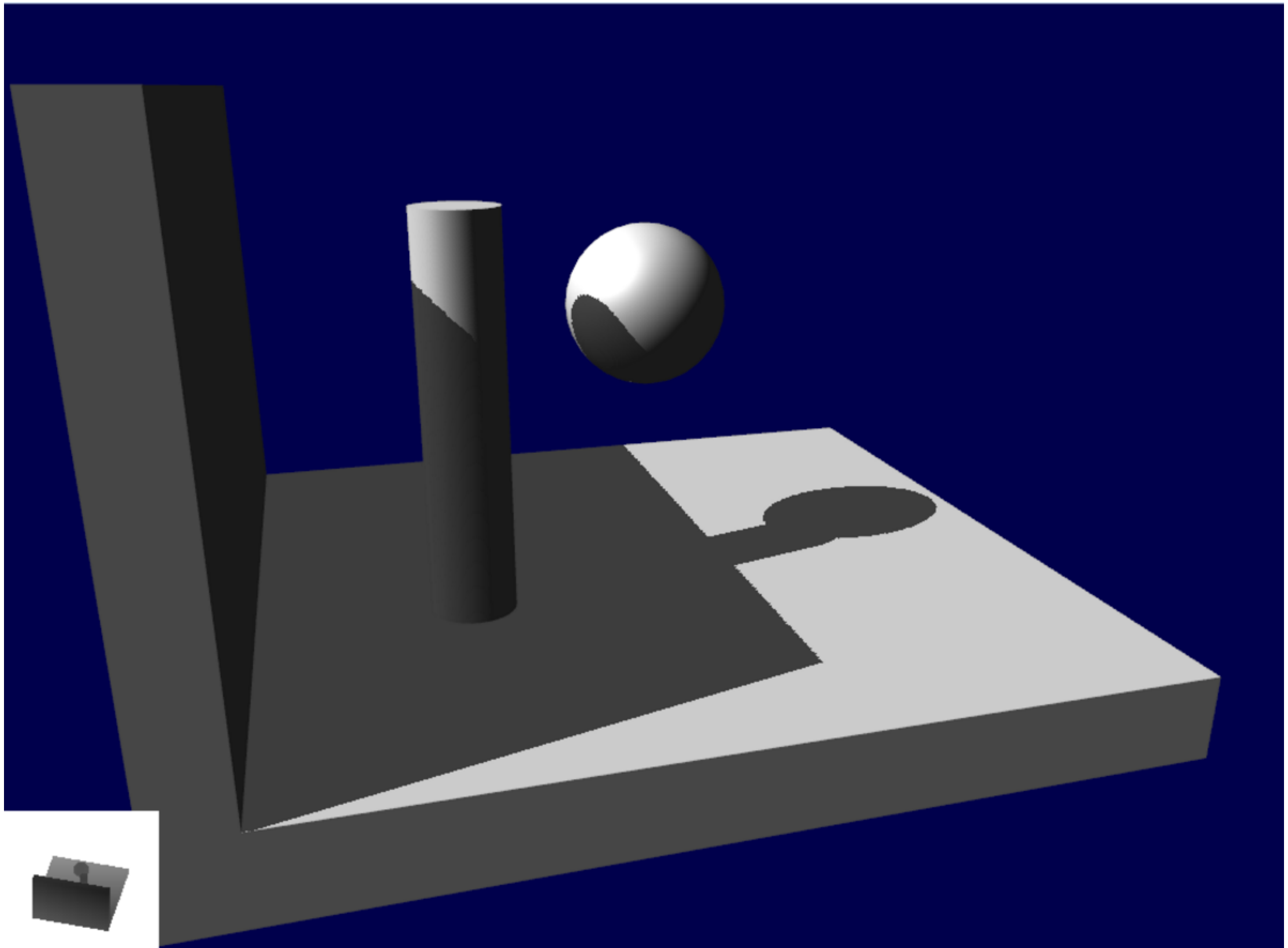
Peter Panning

- Wrong shading of the ground, making the wall to look as if it’s flying (hence the term “Peter Panning”), add bias make it worse



Peter Panning

- Solve: simply avoid thin geometry
- Result:



Other problems: like aliasing
Other tricks: like PCF

You can refer to:

https://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/
https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

Thanks