Introduction to Computer Graphics (CS360A)

Instructor: Soumya Dutta

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur (IITK)

email: soumyad@cse.iitk.ac.in

Acknowledgements

- A subset of the slides that I will present throughout the course are adapted/inspired by excellent courses on Computer Graphics offered by Prof. Han-Wei Shen, Prof. Wojciech Matusik, Prof. Frédo Durand, Prof. Abe Davis, and Prof. Cem Yuksel

Midterm

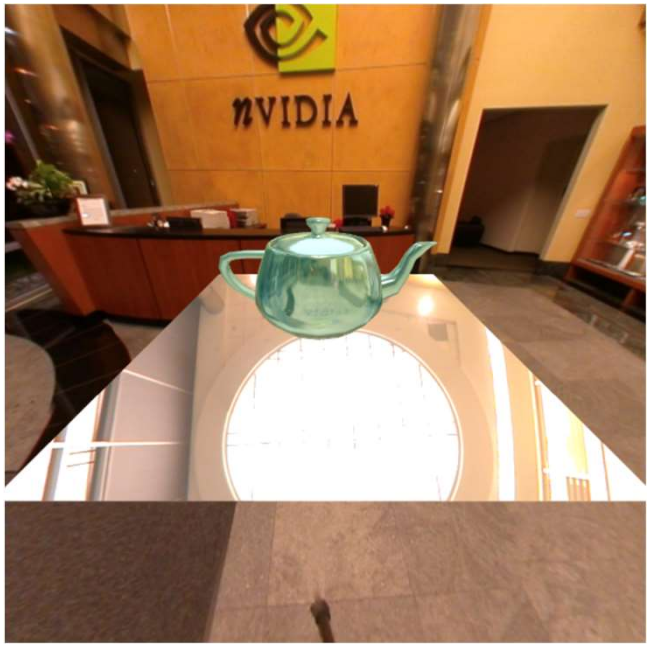
- **Syllabus: Everything up to today's class**

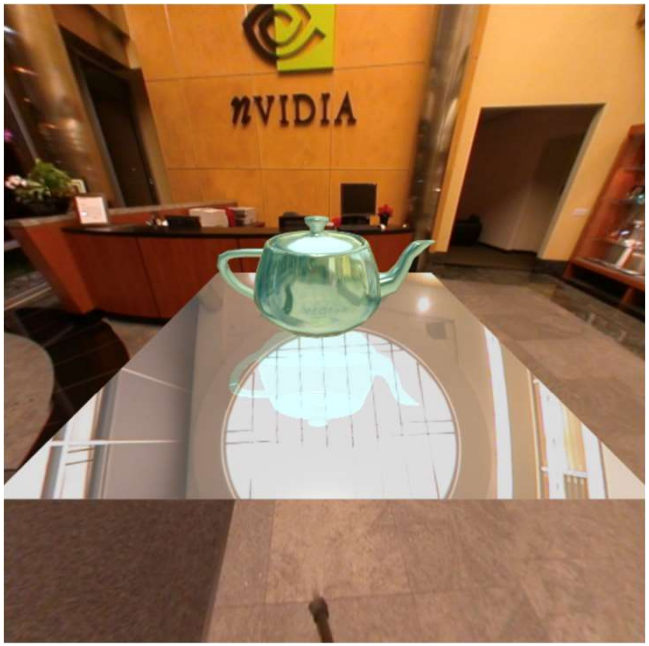
- Test on concepts
  - Various coordinate spaces
  - Transformations, Projections, Shading
  - Matrix representations
  - GPU Pipeline
- Short Subjective type questions
- Code fragments to check your understanding

Last Class: Reflection Mapping

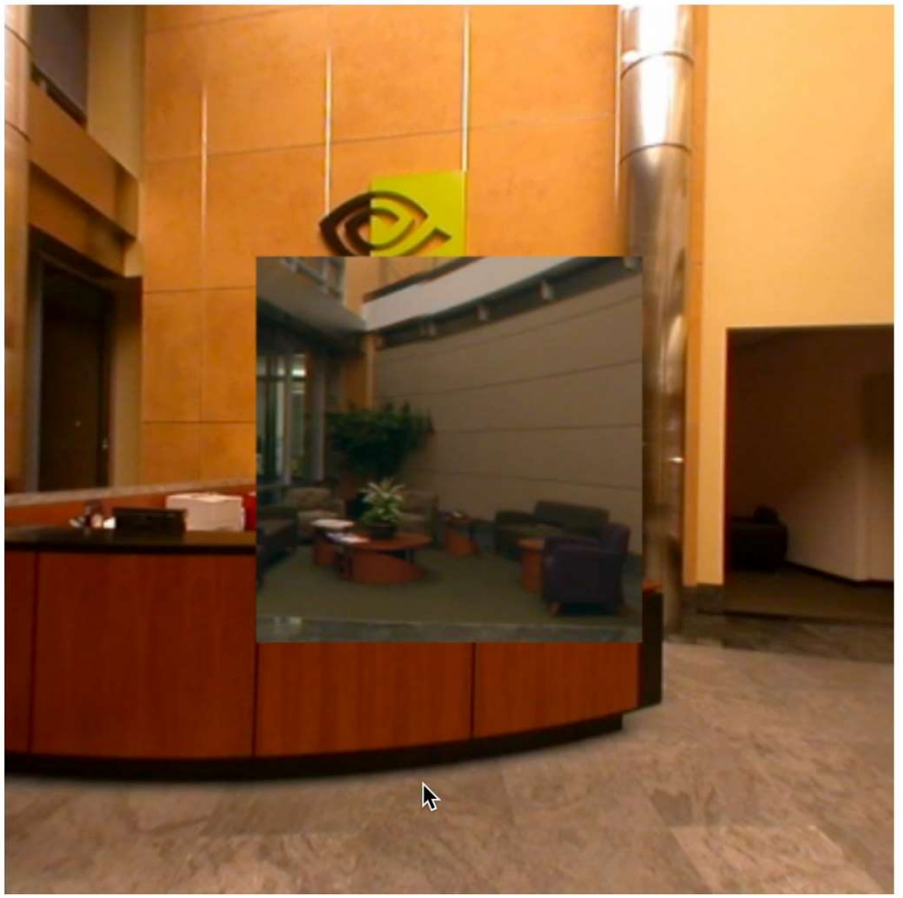
Two Types of Reflection Mapping

- Environment Reflection of Objects
- Object's own reflection on another object/surface



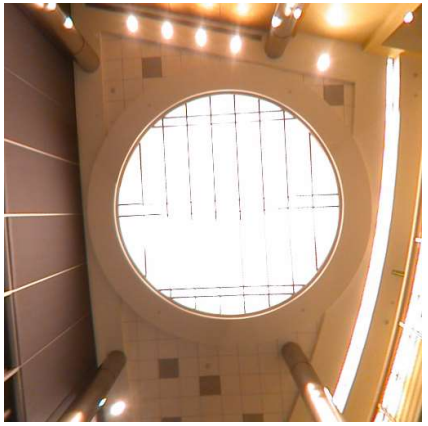
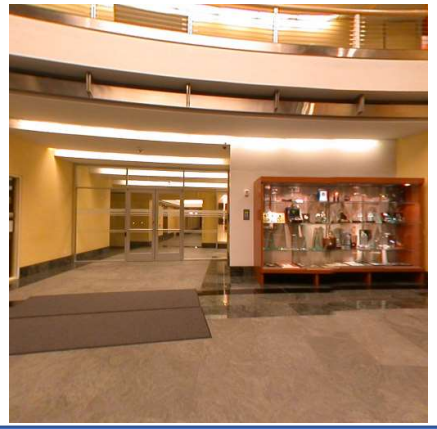


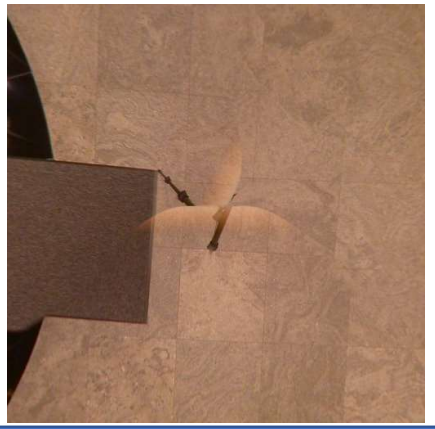
Environment Reflection



Implementation Details

- Load the 6 texture images



Implementation Details: Load Images


```
var cubeMapPath = "CubeMapTextures/Nvidia/";
var posx, posy, posz, negx, negy, negz;

var posx_file = cubeMapPath.concat("posx.jpg");
var posy_file = cubeMapPath.concat("posy.jpg");
var posz_file = cubeMapPath.concat("posz.jpg");
var negx_file = cubeMapPath.concat("negx.jpg");
var negy_file = cubeMapPath.concat("negy.jpg");
var negz_file = cubeMapPath.concat("negz.jpg");
```

Implementation Details: Load Images

- *function webGLStart() {*

  initCubeMap();

  posx = initTextures(posx_file);

  posy = initTextures(posy_file);

  posz = initTextures(posz_file);

  negz = initTextures(negz_file);

  negx = initTextures(negx_file);

  negy = initTextures(negy_file);

  ....

}

Implementation Details: Load Images

- *function initTextures(textureFile) {*

  var tex = gl.createTexture();

  tex.image = new Image();

  tex.image.onload = function () {

  handleTextureLoaded(tex);

  };

  tex.image.src = textureFile;

  return tex;

}

- *handleTextureLoaded(tex)*
  - You have seen this function before, code is provided also in the texturmap example

Implementation Details: initCubeMap()


```
function initCubeMap() {
  const faceInfos = [
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_X,
      url: posx_file,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_X,
      url: negx_file,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Y,
      url: posy_file,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Y,
      url: negy_file,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Z,
      url: posz_file,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Z,
      url: negz_file,
    },
  ];
```

Specifying the mapping for the 6 cube faces

Implementation Details: initCubeMap()


```
cubemapTexture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_CUBE_MAP, cubemapTexture);

faceInfos.forEach((faceInfo) => {
  const { target, url } = faceInfo;
  // setup each face
  gl.texImage2D(
    target,
    0,
    gl.RGBA,
    512,
    512,
    0,
    gl.RGBA,
    gl.UNSIGNED_BYTE,
    null
  );
```

Setup texture specifications for each 6 faces

Implementation Details: initCubeMap()


```
    // load images
    const image = new Image();
    image.src = url;
    image.addEventListener("load", function () {
      gl.bindTexture(gl.TEXTURE_CUBE_MAP, cubemapTexture);
      gl.texImage2D(target, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
      gl.generateMipmap(gl.TEXTURE_CUBE_MAP);
      drawScene();
    });
  });

  // uses mipmap for texturing
  gl.generateMipmap(gl.TEXTURE_CUBE_MAP);
  gl.texParameteri(
    gl.TEXTURE_CUBE_MAP,
    gl.TEXTURE_MIN_FILTER,
    gl.LINEAR_MIPMAP_LINEAR
  );
}
```

Load images and generate mipmap

Implementation Details: Shaders

- JS: Create Model → World Transformation Matrix for Normals

wnMatrix = mat4.transpose(mat4.inverse(mMatrix));

- Vertex Shader:

```
uniform mat4 uWNMatrix;

out vec3 v_worldPosition;
out vec3 v_worldNormal;

//Computation is done at world space
v_worldPosition = mat3(uMMatrix) * aPosition;
v_worldNormal = mat3(uWNMatrix) * aNormal;
```

Implementation Details: Shaders

- Fragment Shader:

```
uniform samplerCube cubeMap;
uniform vec3 eyePos; // in world space, sent from JS
in vec3 v_worldPosition;
in vec3 v_worldNormal;

// cubemap color calculation
vec3 worldNormal = normalize(v_worldNormal);
vec3 eyeToSurfaceDir = normalize(v_worldPosition - eyePos);
```

Implementation Details: Shaders

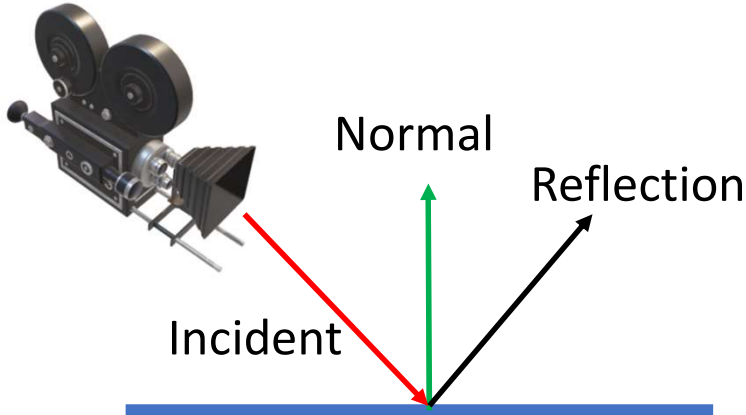
- Fragment Shader:

// calculate reflection vector

vec3 directionReflection = reflect(eyeToSurfaceDir,worldNormal);

vec4 cubeMapReflectCol = texture(cubeMap, directionReflection);

fragColor = cubeMapReflectCol;



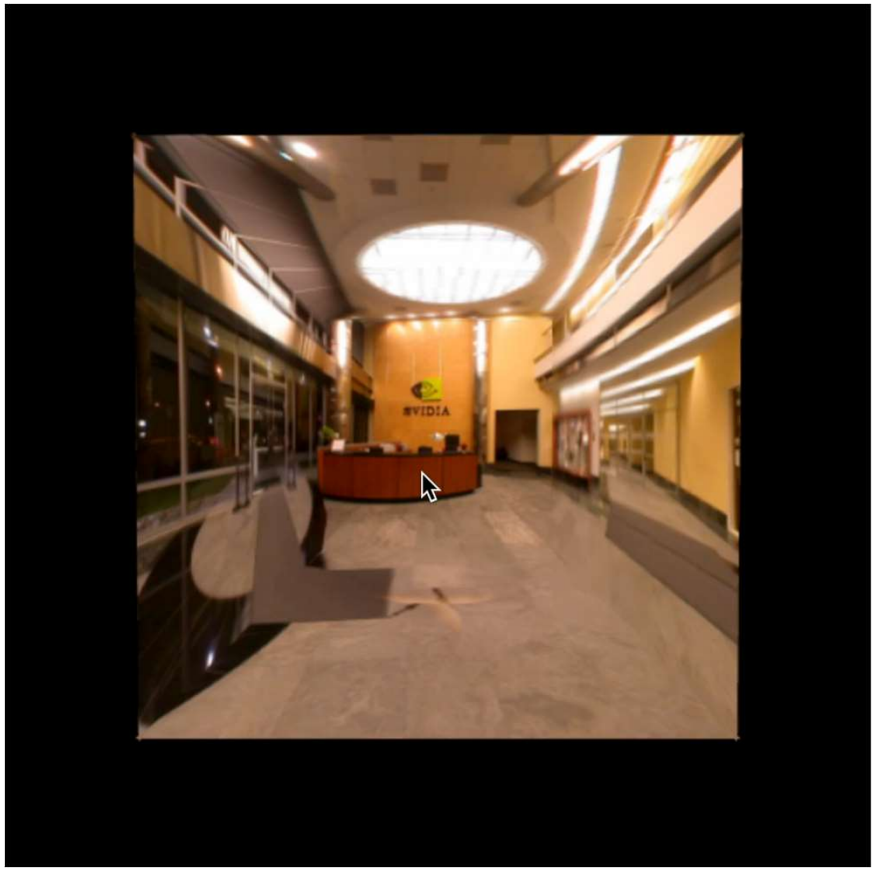
reflect function in GLSL assumes that the incident direction is from camera/eye to object surface

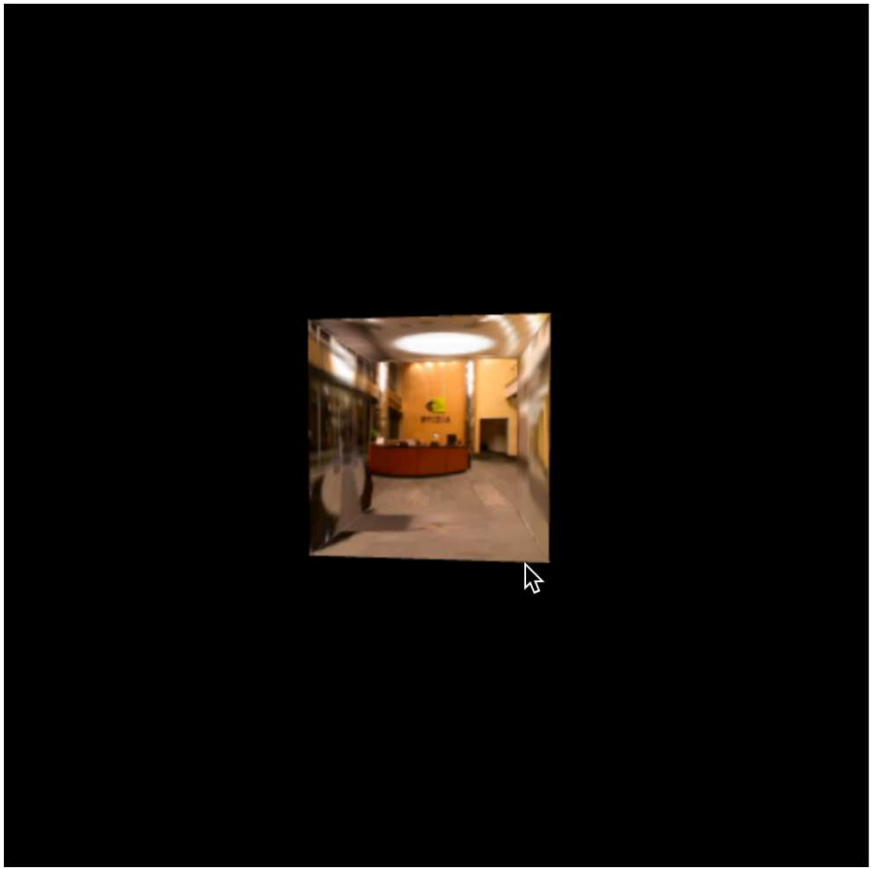
SkyBox


```
drawSkyBox(){

    // Back side of the cube
    pushMatrix(matrixStack, mMatrix);

    // texture setup for use
    gl.activeTexture(gl.TEXTURE1); // set texture unit 1 to use
    gl.bindTexture(gl.TEXTURE_2D, negz); // bind the texture object
    gl.uniform1i(uRegTexLocation, 1); // pass the texture unit

    // transformations
    mMatrix = mat4.translate(mMatrix, [0, 0, -99.5]);
    mMatrix = mat4.rotate(mMatrix, degToRad(180), [0, 0, 1]);
    mMatrix = mat4.scale(mMatrix, [200, 200, 200]);

    color = [0.0, 1.0, 1.0, 1.0];
    drawSquare(color);

    mMatrix = popMatrix(matrixStack);

    // Front side of the cube
    ........
}
```

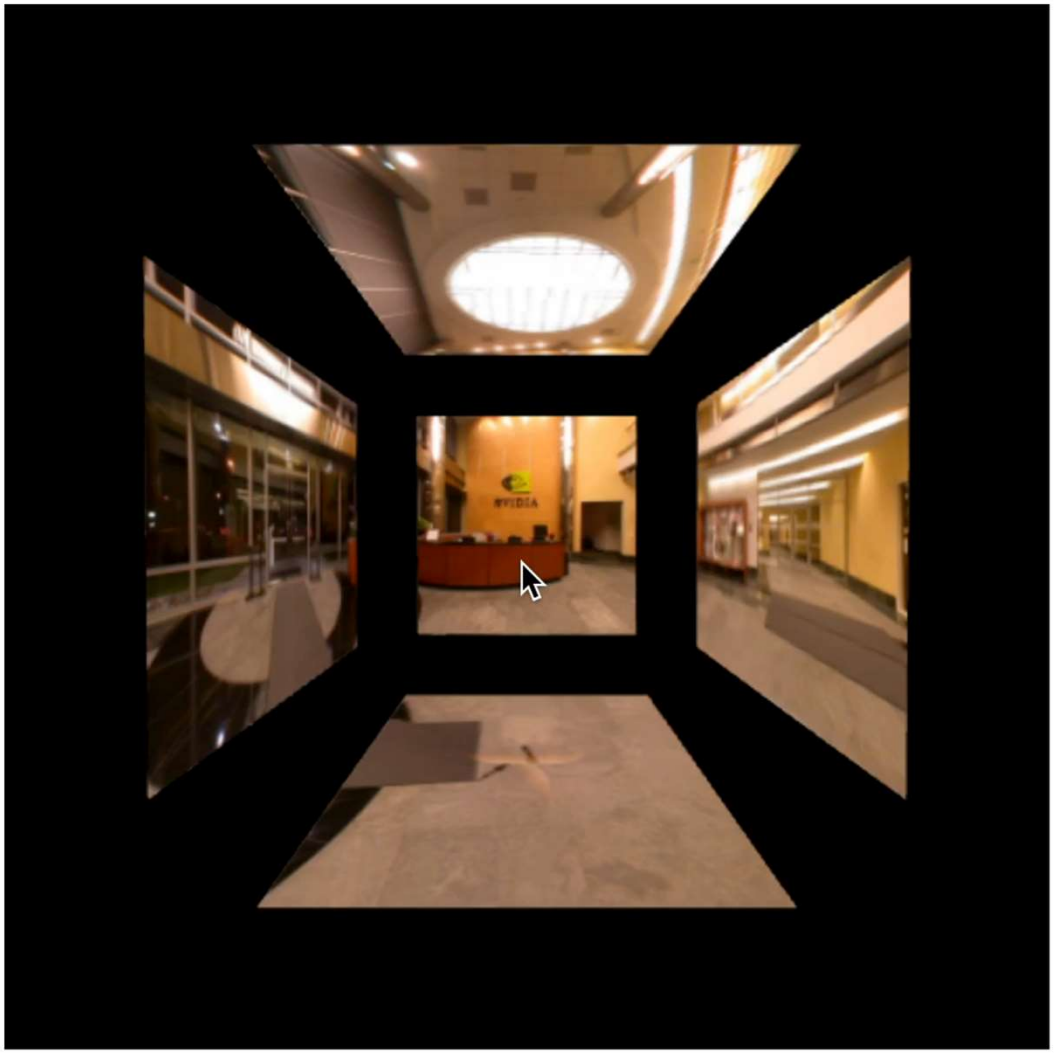
This color does not matter in this case since we will use color from texture

SkyBox





SkyBox



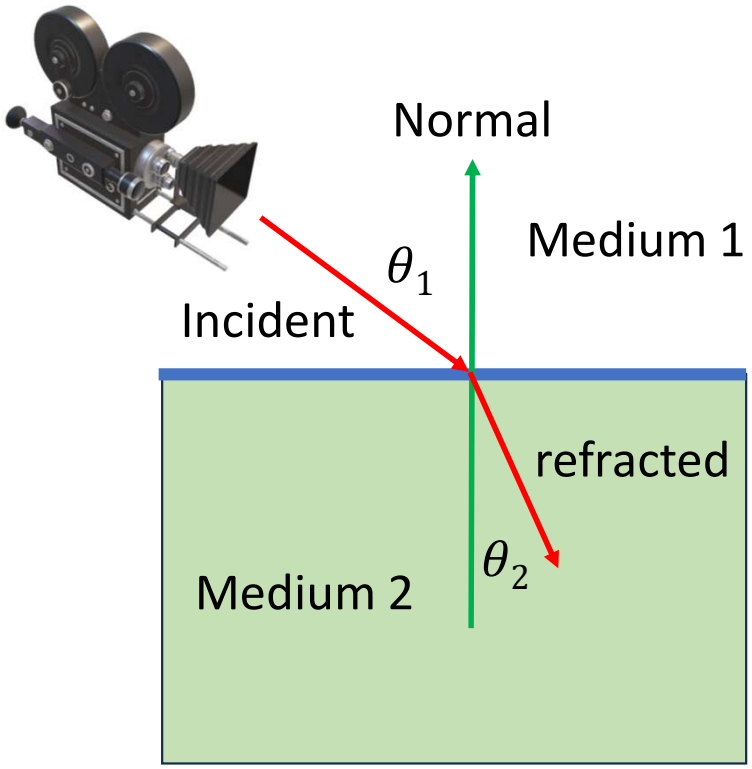
How About Refraction?

- Objects that allow light to pass through
- Light bends when going from one medium (air) to another medium (glass)
- Snell's law:

$$\frac{\sin(\theta_1)}{\sin(\theta_2)} = \frac{\eta_2}{\eta_1} = \eta_{21}$$

η_1 = refractive index of Medium 1

η_2 = refractive index of Medium 2



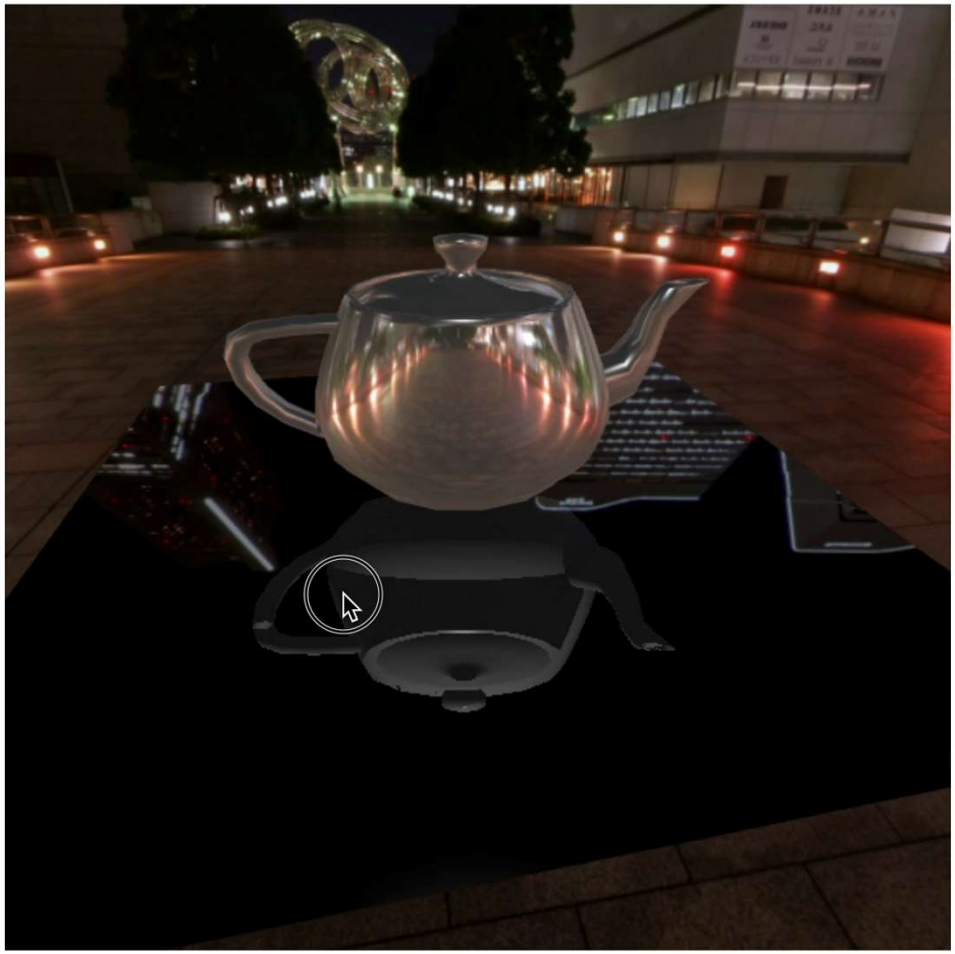
Refraction

Easy to Do It In Shader with CubeMap

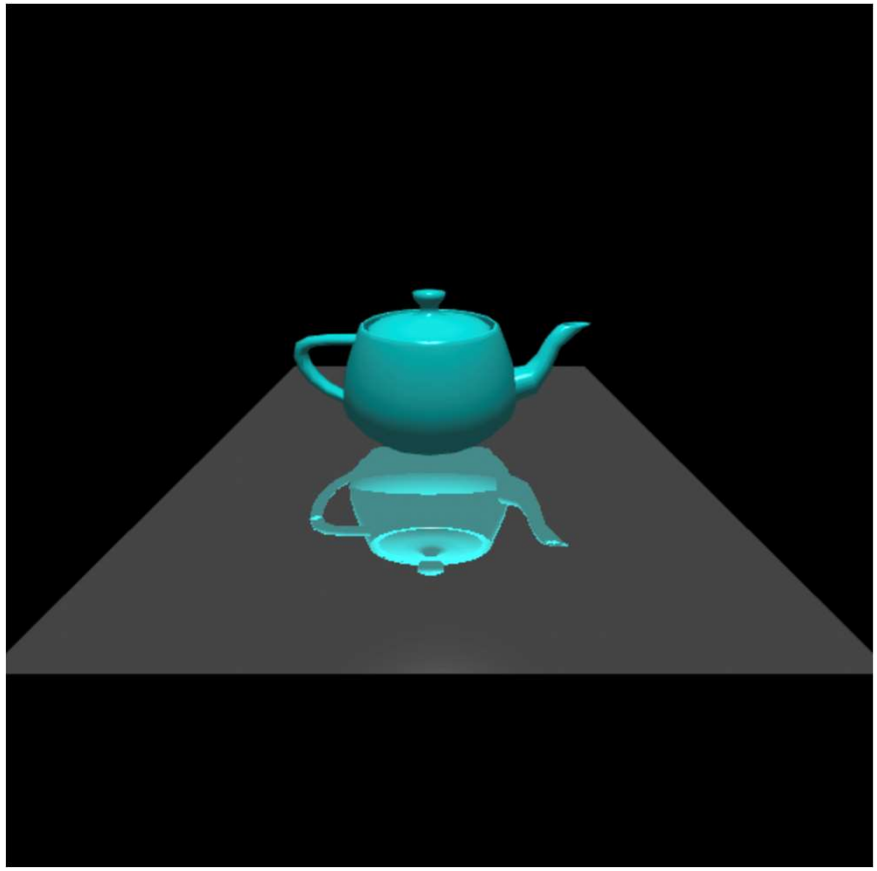
- We use the GLSL function refract to get the direction of refracted ray
- Use refracted ray to perform cubemap texture lookup
- That's it!

```
// calculate direction cubemap refraction
vec3 directionRefraction = refract(eyeToSurfaceDir,worldNormal,eta);
vec4 cubeMapRefractCol = texture(cubeMap, directionRefraction);
```

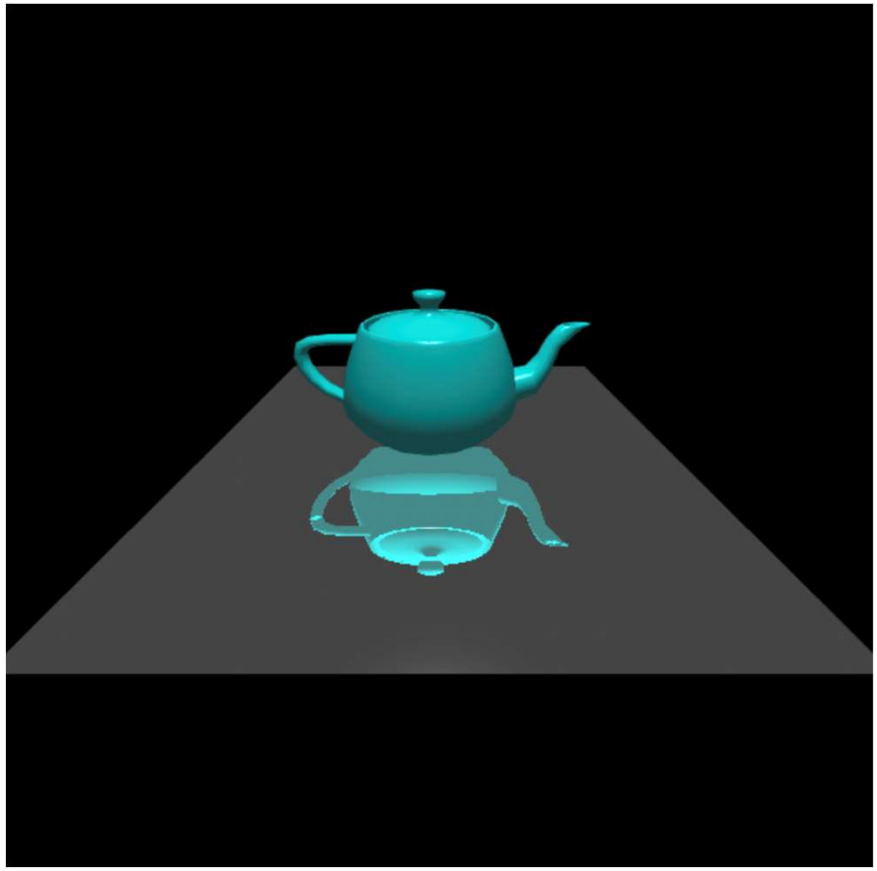
Planer Reflection

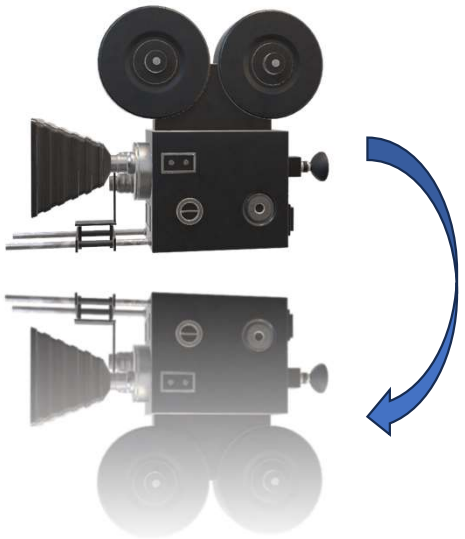


Planar Reflection

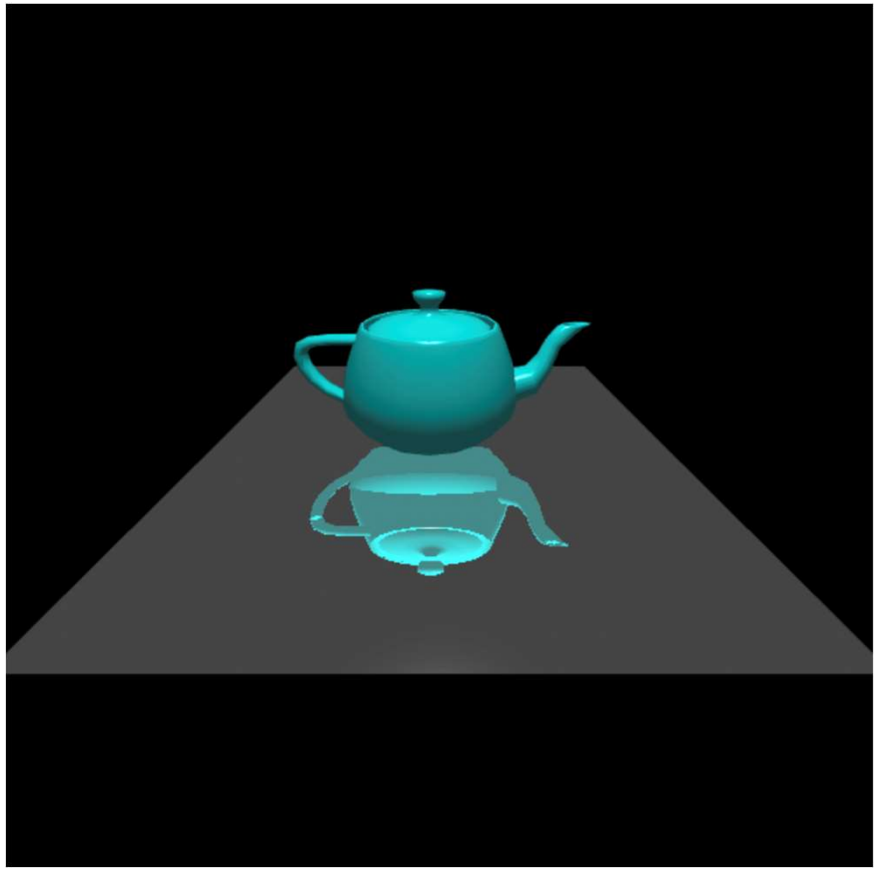


Planar Reflection



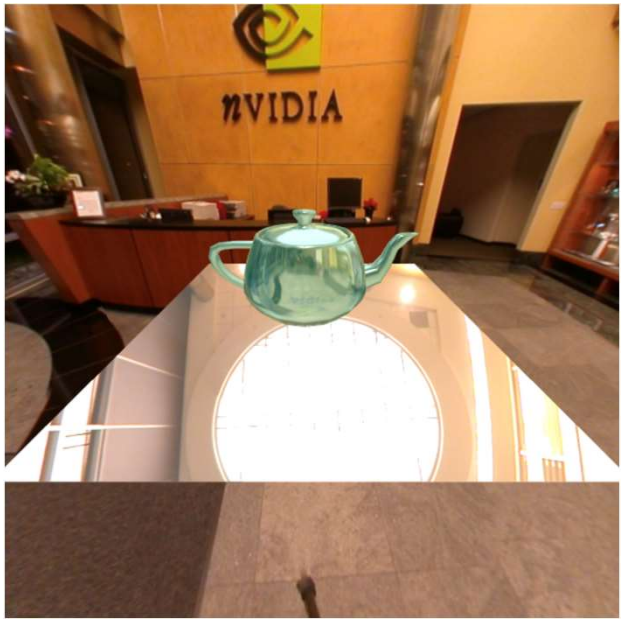


Planar Reflection



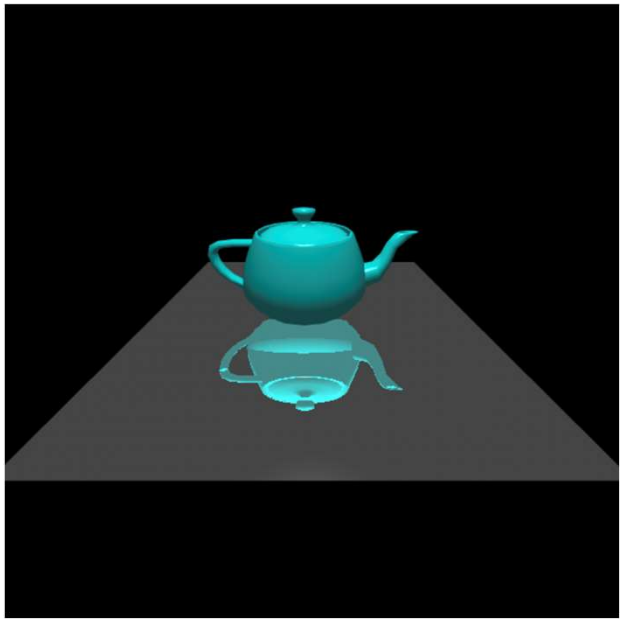
- Two pass rendering: Render the scene twice
- 1. Flip the camera and render the object flipped and store it into a texture (FBO)
- 2. Then during second pass, use the stored image as texture to add the reflection

Planar Reflection: Conceptual Idea



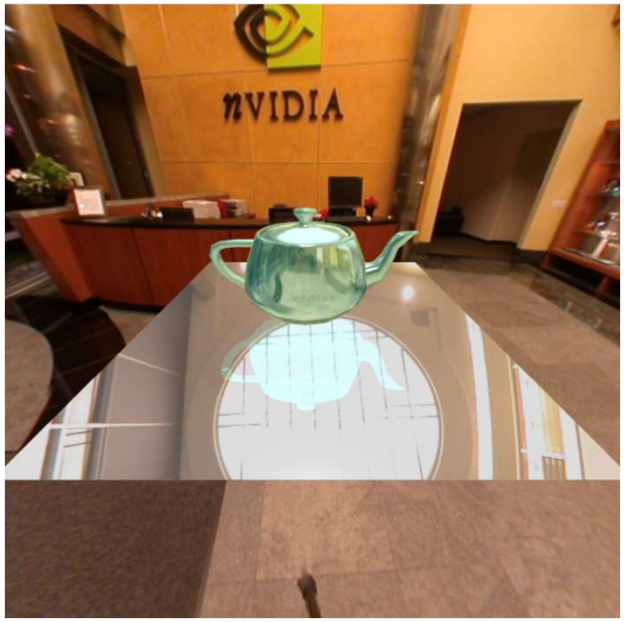
Scene with no reflection texture

+



Framebuffer (reflection) texture

=

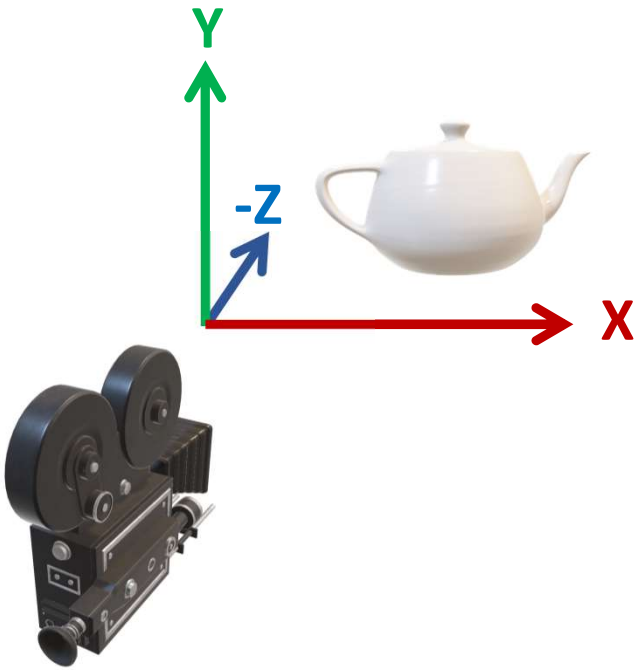


Final rendered image with planar and cubemap reflection

Frame Buffer Objects For Planar Reflection

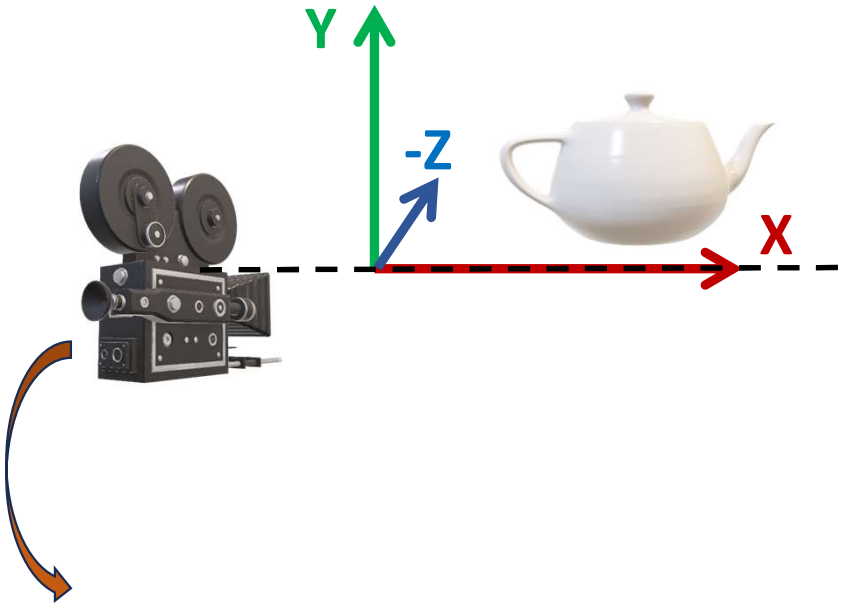
- Pass 1: Render the object that is supposed to produce reflection inverted and store the result into a FBO texture
  - Also known as 'render to texture'
  - This inverted image can be generated by flipping your camera and taking an image of the object
- Pass 2: Now, render the image to the default framebuffer, i.e., on the screen/canvas so that we can see it
  - Here, we use the rendered image from Pass 1 as a texture map
  - Blend the color from Pass 2 with Pass 1 to get the inverted object
  - Create the illusion of 'reflection'

How Do We Invert/Flip Camera in Pass 1?



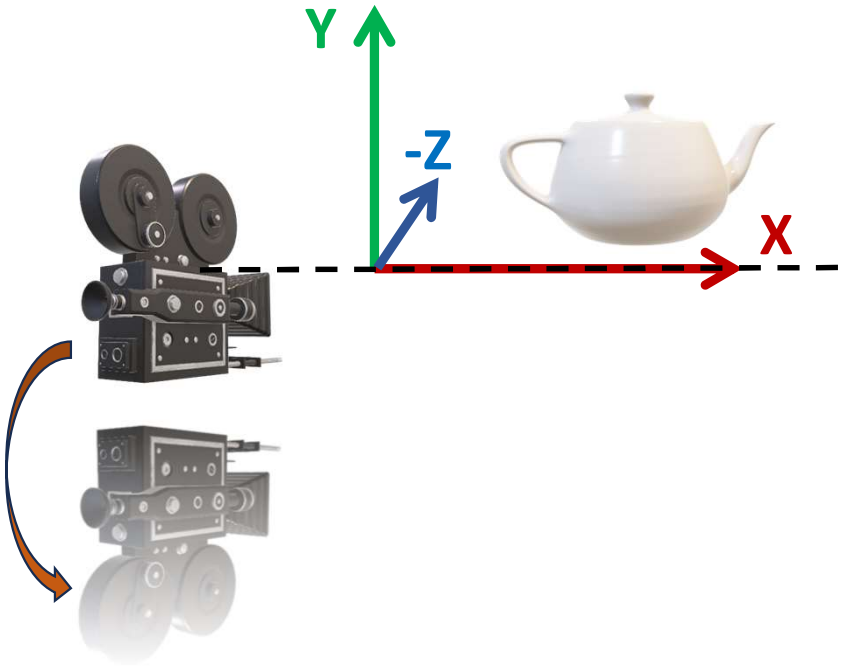
- What are we trying to achieve?
- Flip everything with respect to the X-Z plane

How Do We Invert/Flip Camera in Pass 1?



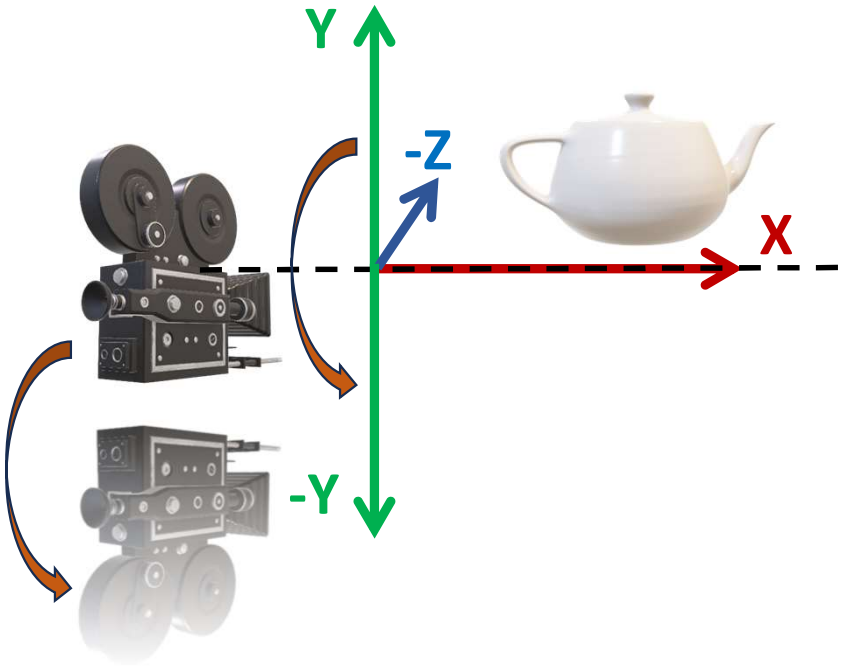
- What are we trying to achieve?
- Flip everything with respect to the X-Z plane
  - How?

How Do We Invert/Flip Camera in Pass 1?



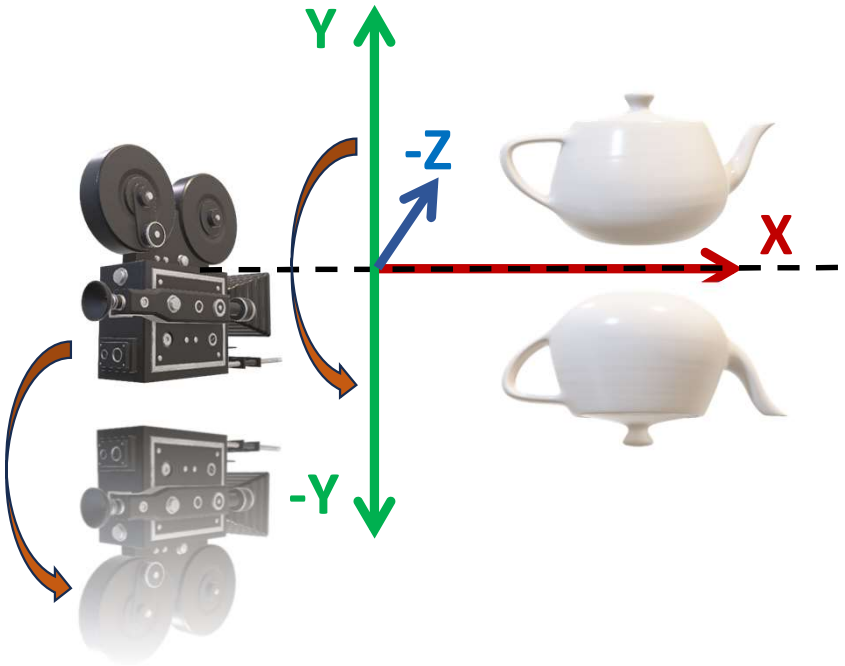
- What are we trying to achieve?
- Flip everything with respect to the X-Z plane
  - How?

How Do We Invert/Flip Camera in Pass 1?



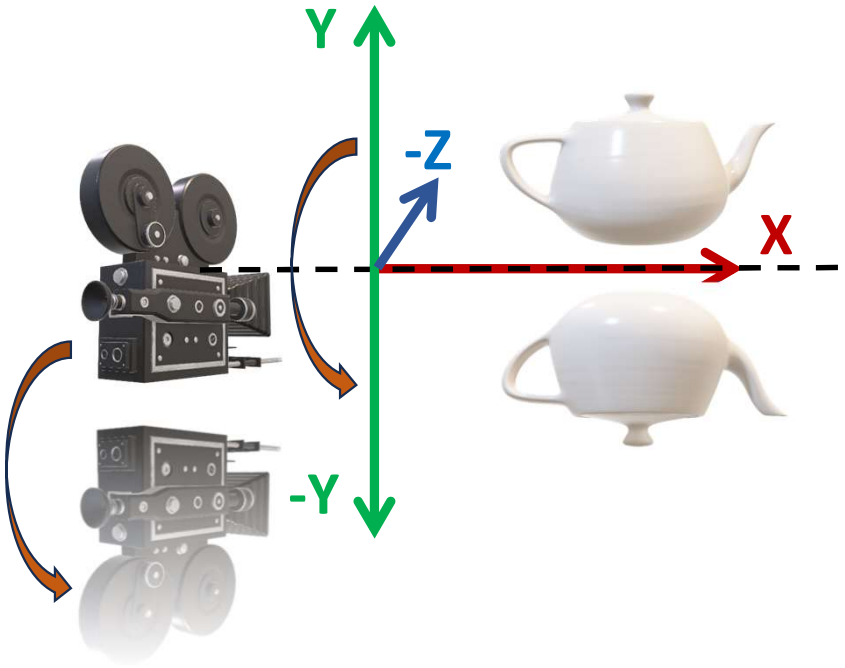
- What are we trying to achieve?
- Flip everything with respect to the X-Z plane
  - How?

How Do We Invert/Flip Camera in Pass 1?



- What are we trying to achieve?
- Flip everything with respect to the X-Z plane
  - How?

How Do We Invert/Flip Camera in Pass 1?



- What are we trying to achieve?
- Flip everything with respect to the X-Z plane
  - How?
- Multiply the Y coordinates of all points with -1!
- Scale the View matrix with [1,-1,1] during the first pass rendering and we are done!
  - ***ViewMat = ScaleMat[1,-1,1]*ViewMat***