Introduction to Computer Graphics (CS360A)

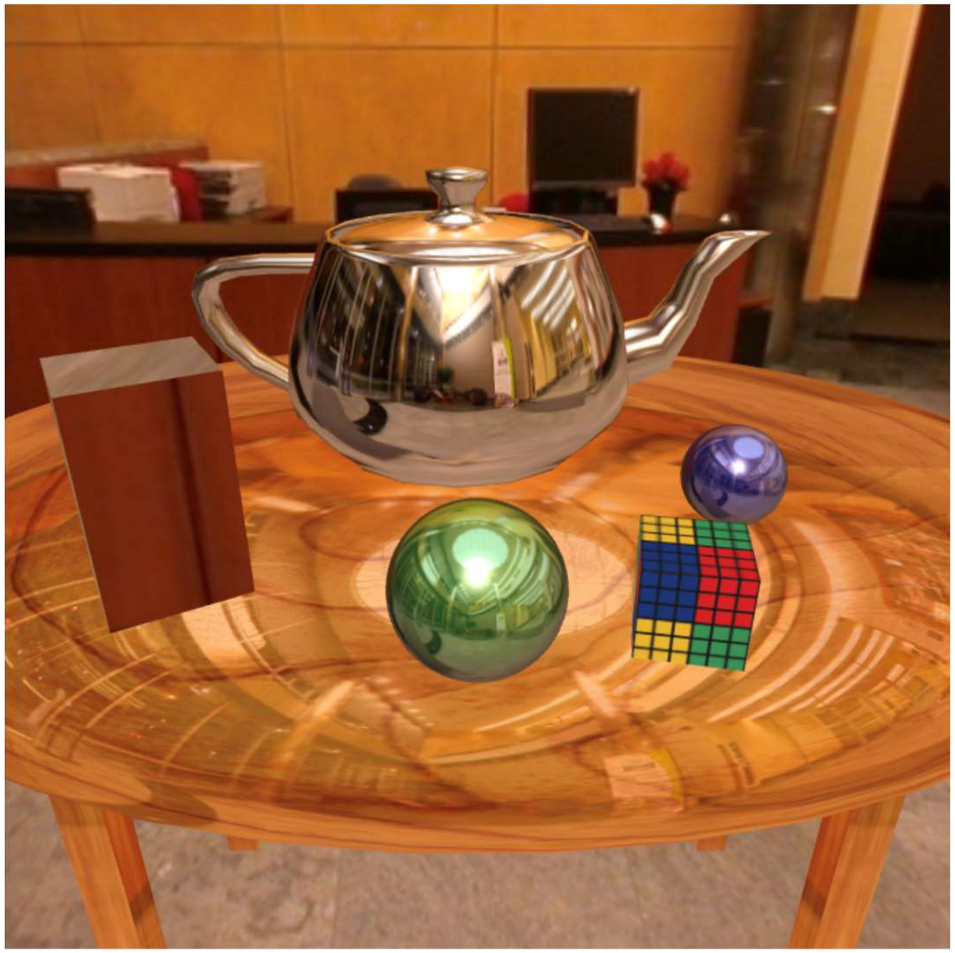
Instructor: Soumya Dutta

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur (IITK)

email: soumyad@cse.iitk.ac.in

Assignment 3: Due Oct 7, 11:59pm



Discarding Fragments

- We often want to create objects that have holes in them
- We may also want to discard some fragment colors
- Solution: Selectively discard certain fragment colors during shader code execution in GPU
  - Achieved using texture mapping
  - Use textures that have alpha channel
  - If alpha channel is transparent then “discard” the fragment

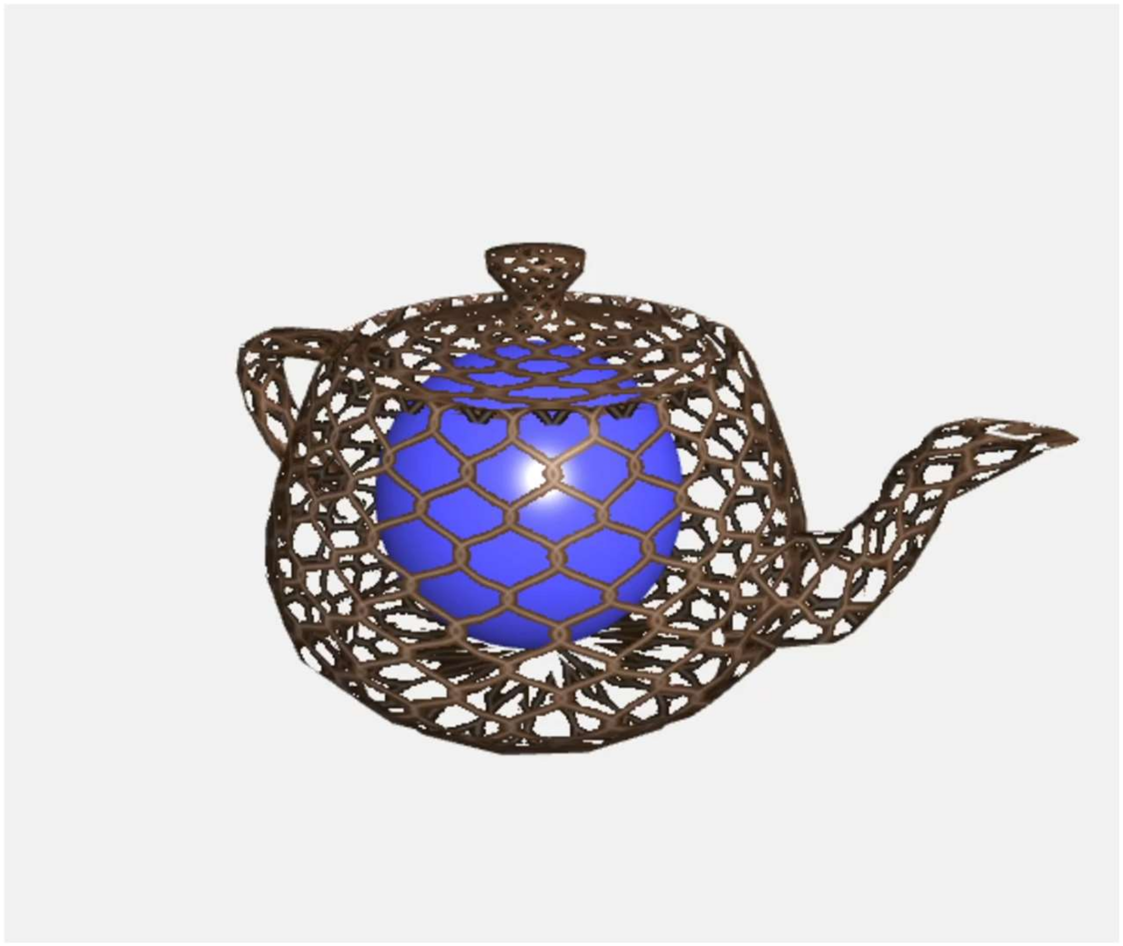
This texture has alpha channel info

Texture-based Fragment Discarding

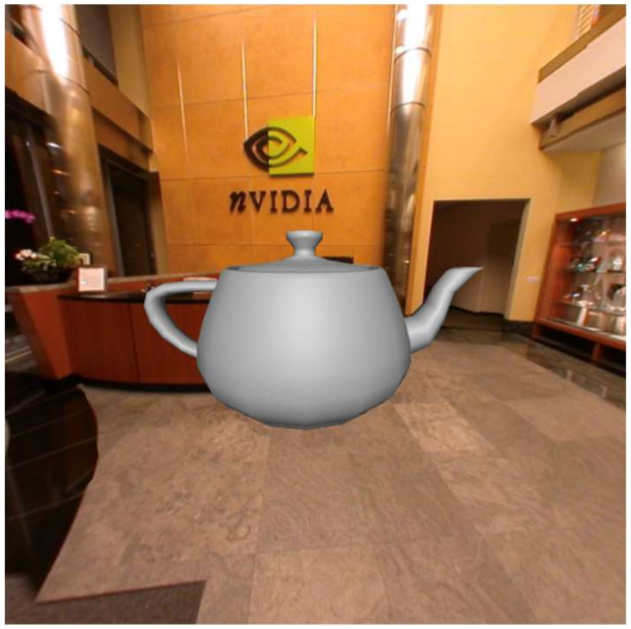
```
// When setting up the texture
gl.texImage2D(
      gl.TEXTURE_2D, 0,
      gl.RGBA, gl.RGBA,
      gl.UNSIGNED_BYTE, texture.image);

// In fragment shader
vec4 textureColor = texture(uTexture, fragTexCoord);
if(textureColor.a <= 0.01)
   discard;
else
   fragColor = textureColor;
```

Texture-based Fragment Discarding in 2D

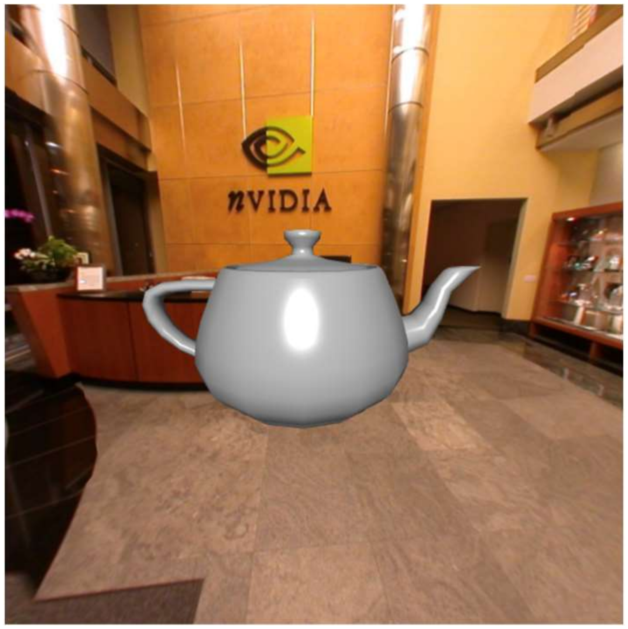
Texture-based Fragment Discarding in 3D



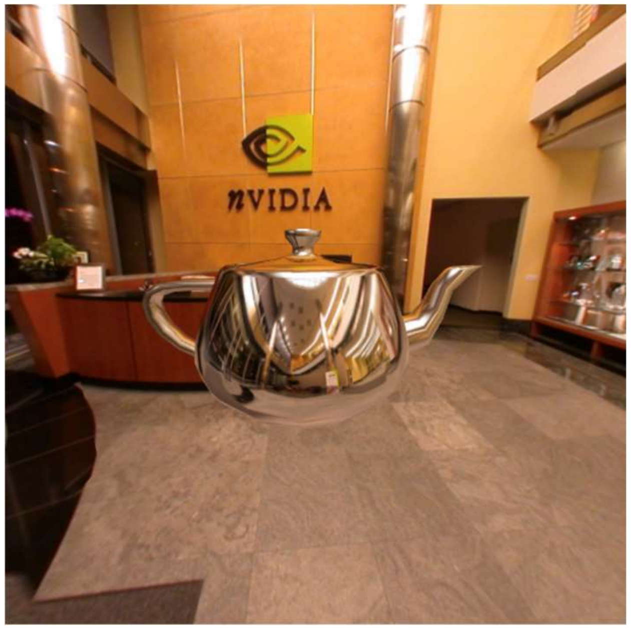
Shading/Illumination Methods So Far



Diffuse object



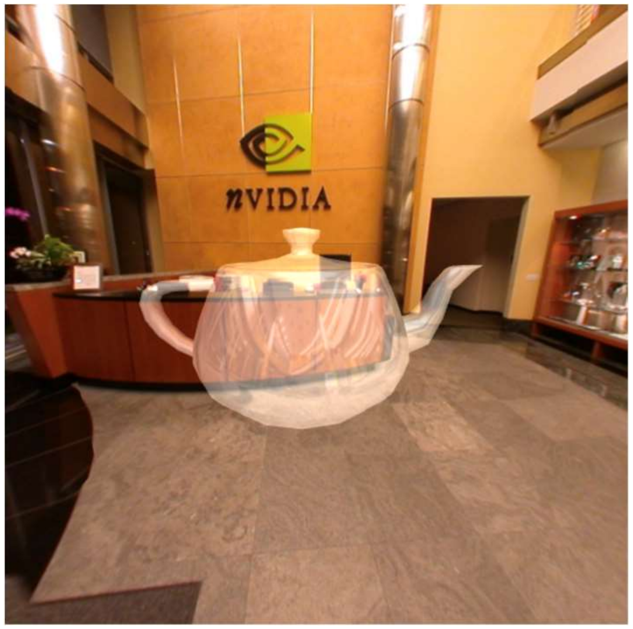
Shiny object (Phong shading)



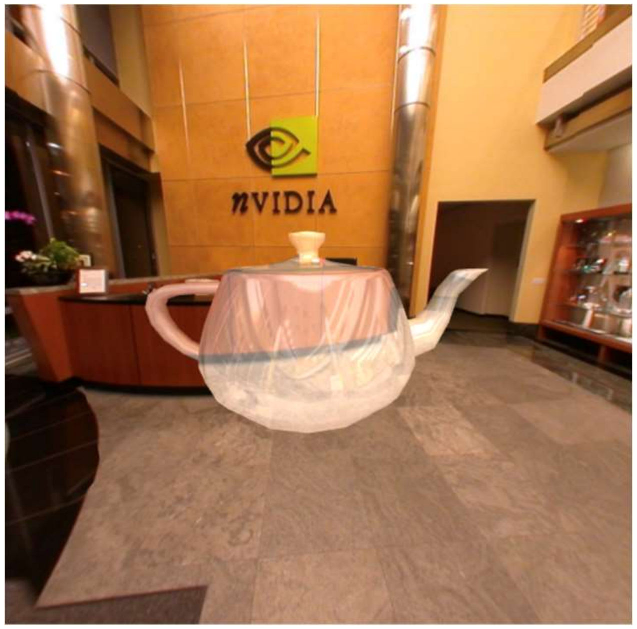
Perfectly Shiny
(Cube-Map reflection)

Advanced Shading Techniques

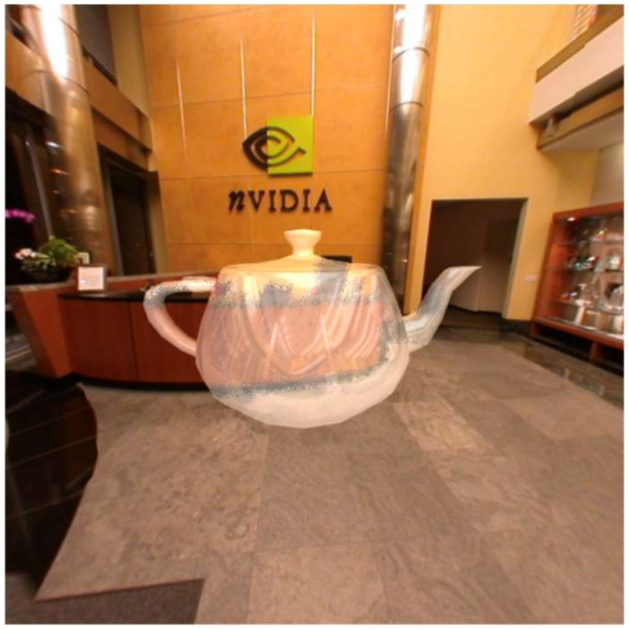
IITK CS360A



See through material



Lens-like material

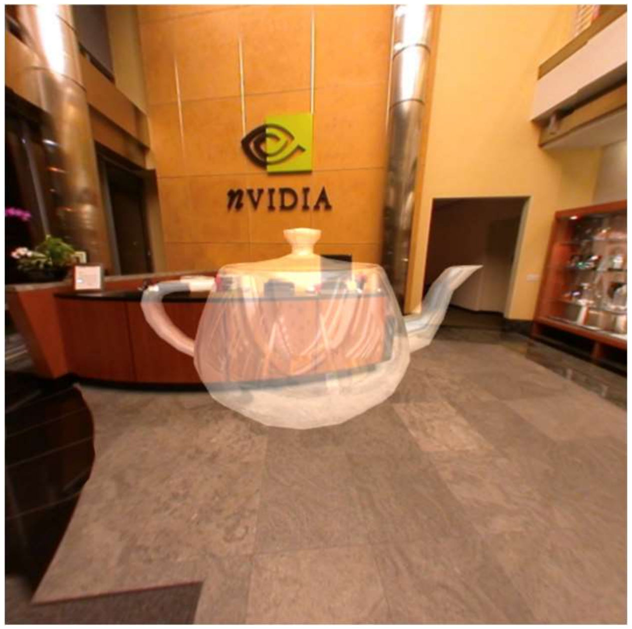


Glossy material

How Do We Produce Such Effects?

- Use FBOs and multi-pass rendering technique intelligently to produce illusion of various advanced shading effects

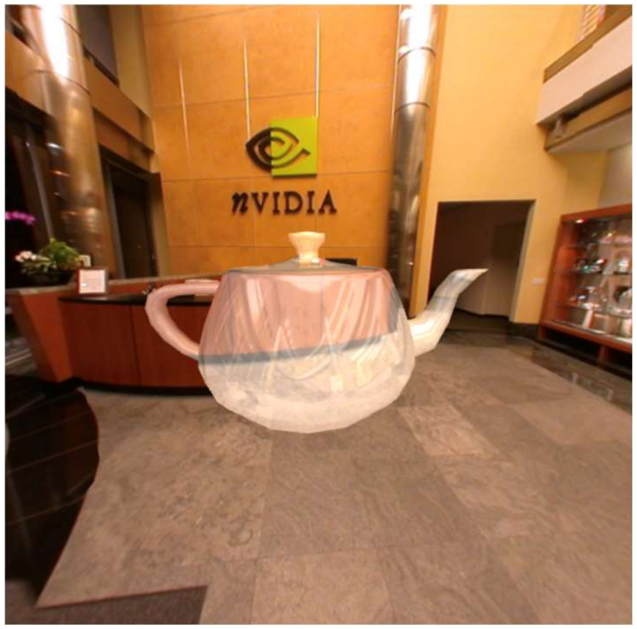
- See-through material:
  - Color on the object is same as the background with some added lighting effects such as reflections



How Do We Produce Such Effects?

- Use FBOs and multi-pass rendering intelligently to produce illusion of such effects in computer graphics

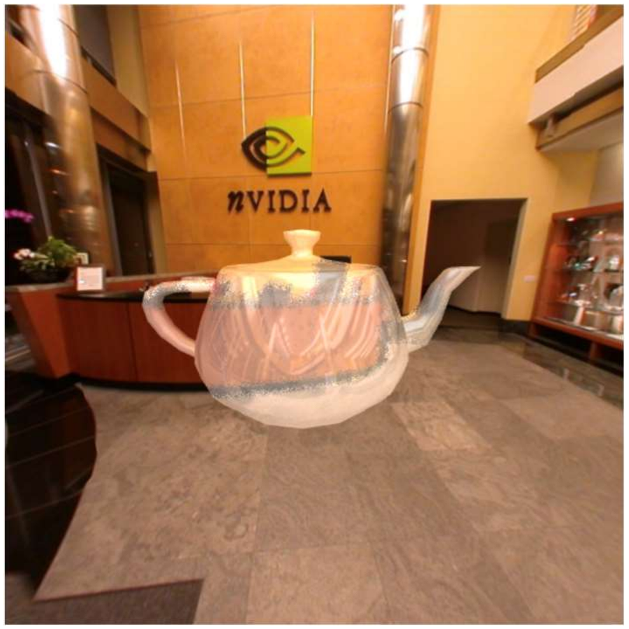
- Lens like material :
  - Color on the object distorts the background fragment color in a way such that it looks like the light is getting bent when passing through the object



How Do We Produce Such Effects?

- Use FBOs and multi-pass rendering intelligently to produce illusion of such effects in computer graphics

- Glossy see-through material :
  - Object shows the background color, but it appears a little noisy/glossy

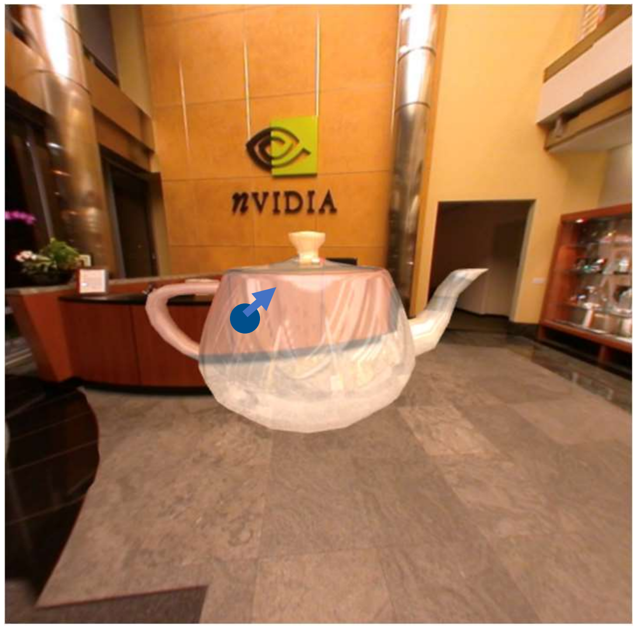


See-through Material Modeling

- We need to know the background color for each fragment on the object
- Then blend the background color with objects own color
- Add reflections or other effects if necessary

Lens-like Material Modeling

- We need to know the background color for each fragment on the object
- Instead of looking up the background color that is right behind the pixel of the object, we look up a fragment that is slightly shifted in x/y direction to artificially create a lens like effect
- Then blend the background color with objects own color
- Add reflections or other effects if necessary

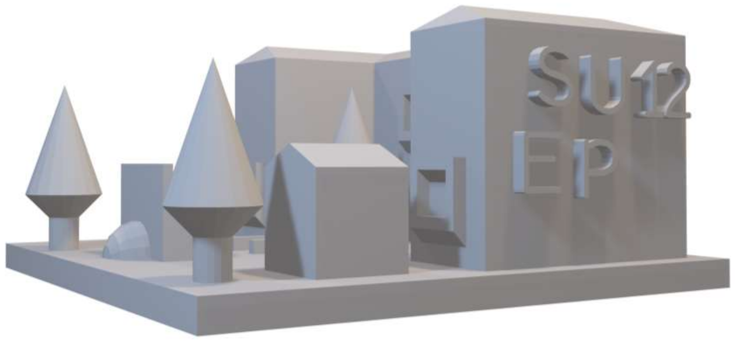


Glossy Material Modeling

- We need to know the background color for each fragment on the object
- Instead of looking up the background color that is right behind the pixel of the object, we "jitter" the location of the fragment slightly and then perform color lookup
- Then blend the background color with objects own color
- Add reflections or other effects if necessary

How Do We Look Up the Background Color?

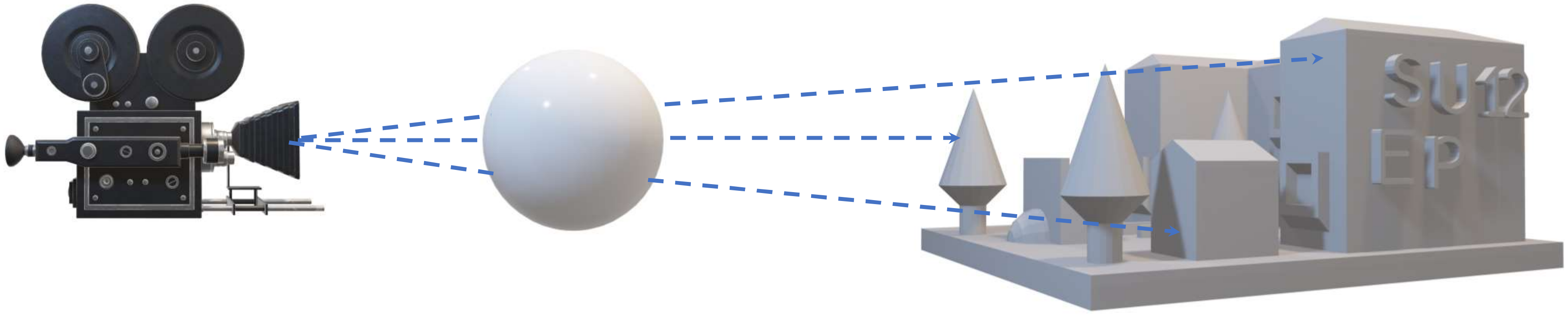
First Pass rendering



- Render the background scene without any objects into an FBO texture buffer during the first pass of the rendering

How Do We Look Up the Background Color?

Second Pass rendering



- Render the background scene without any objects into an FBO texture buffer during the first pass of the rendering
- During the second rendering pass, look up the FBO texture for each fragment on the object to get the background color, i.e., color exactly from behind

How to Look Up FBO Color Using Fragment Texture Coordinates?

IITK CS360A

- We are looking up fragment colors in the “screen space” coordinate
- Fragment shader provides resolution dependent screen space coordinates for each fragment into gl_FragCoord variable
- Normalize gl_FragCoord by dividing it with viewport size to get the look up coordinates in the range (0,1) so that they can be used as texture coordinate
- Use these texture coordinates for FBO texture look up

```
vec2 coord = gl_FragCoord.xy /vec2(canvasSize,canvasSize);
```

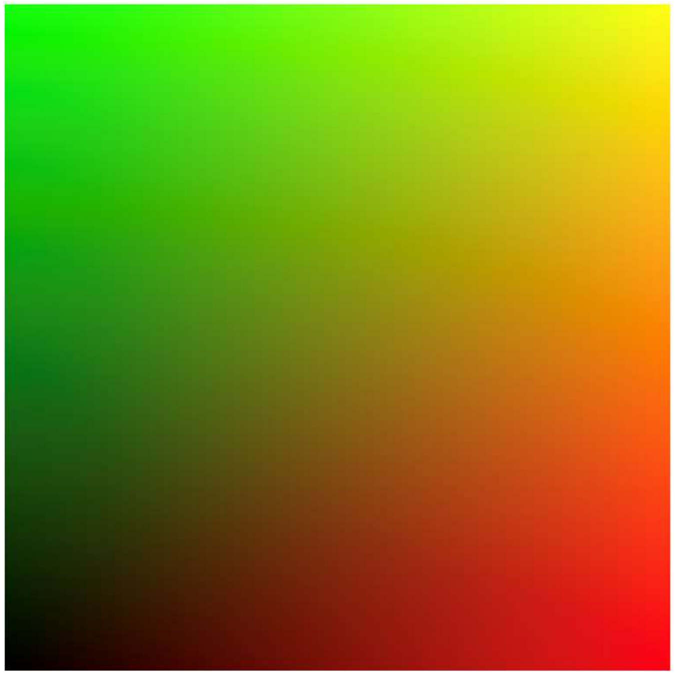
$\uparrow$

Use this coordinate as texture coordinate to lookup from FBO texture

Screen Space Fragment Coordinate

```
vec2 coord = gl_FragCoord.xy / vec2(canvasSize,canvasSize);
fragColor = vec4(coord,0.0,1.0);
```

- Window-relative coordinates of the current fragment
- gl_FragCoord assumes a lower-left origin for window coordinates



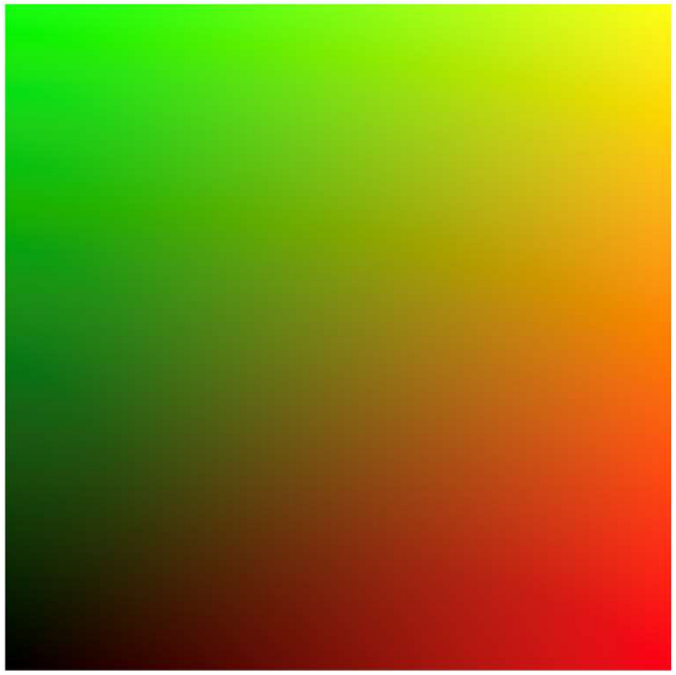
Why we see such a color?

Screen Space Fragment Coordinate

```
vec2 coord = gl_FragCoord.xy / vec2(canvasSize,canvasSize);
fragColor = vec4(coord,0.0,1.0);
```

0,1,0,1=Green

1,1,0,1=Yellow

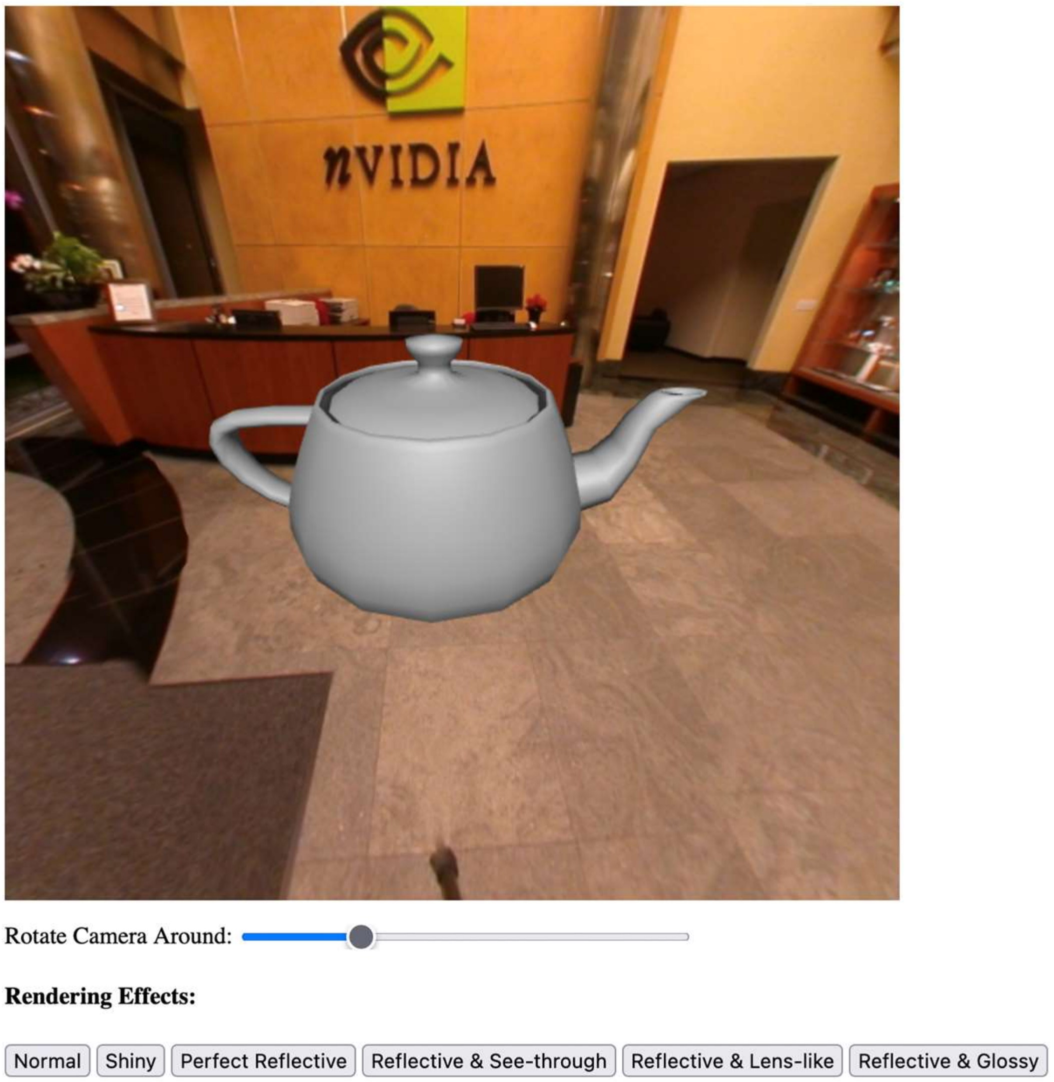


We rendered texture coordinates as colors!

0,0,0,1=Black

1,0,0,1=Red

Demo



Rotate Camera Around:

Rendering Effects: