CSc 165
Computer Game Architecture

09 - Game World:
textures, skyboxes, etc.

Overview

- Texture Mapping
- Game World Backgrounds
- SkyBoxes & SkyDomes
- World Bounds and Visibility
- Render States

Texture Mapping

Basic idea: attach an “image” to an “object”

- Object == polygon(s)
- Images used this way are called *textures*

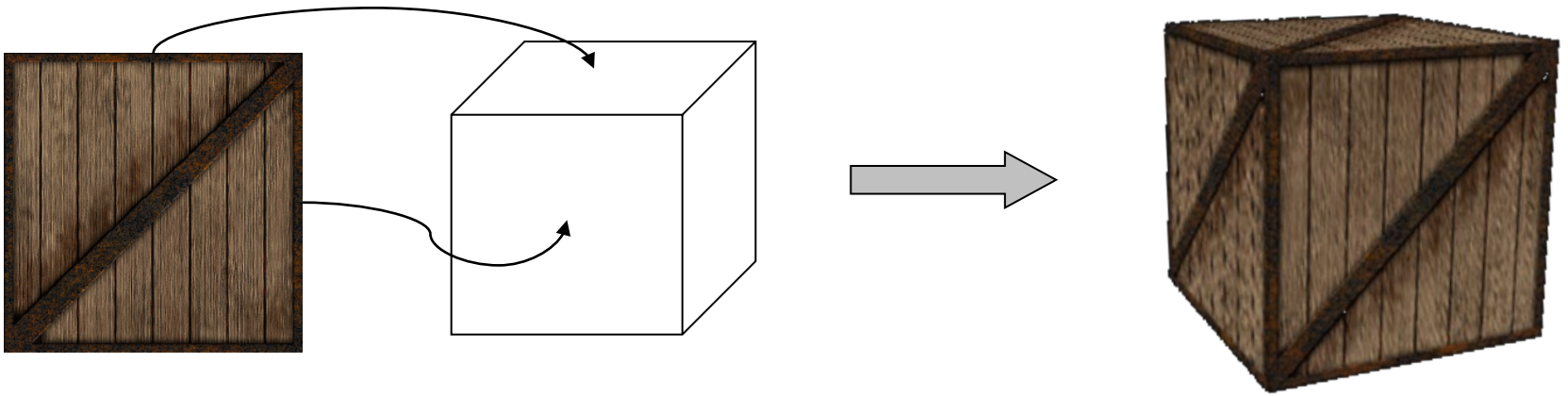
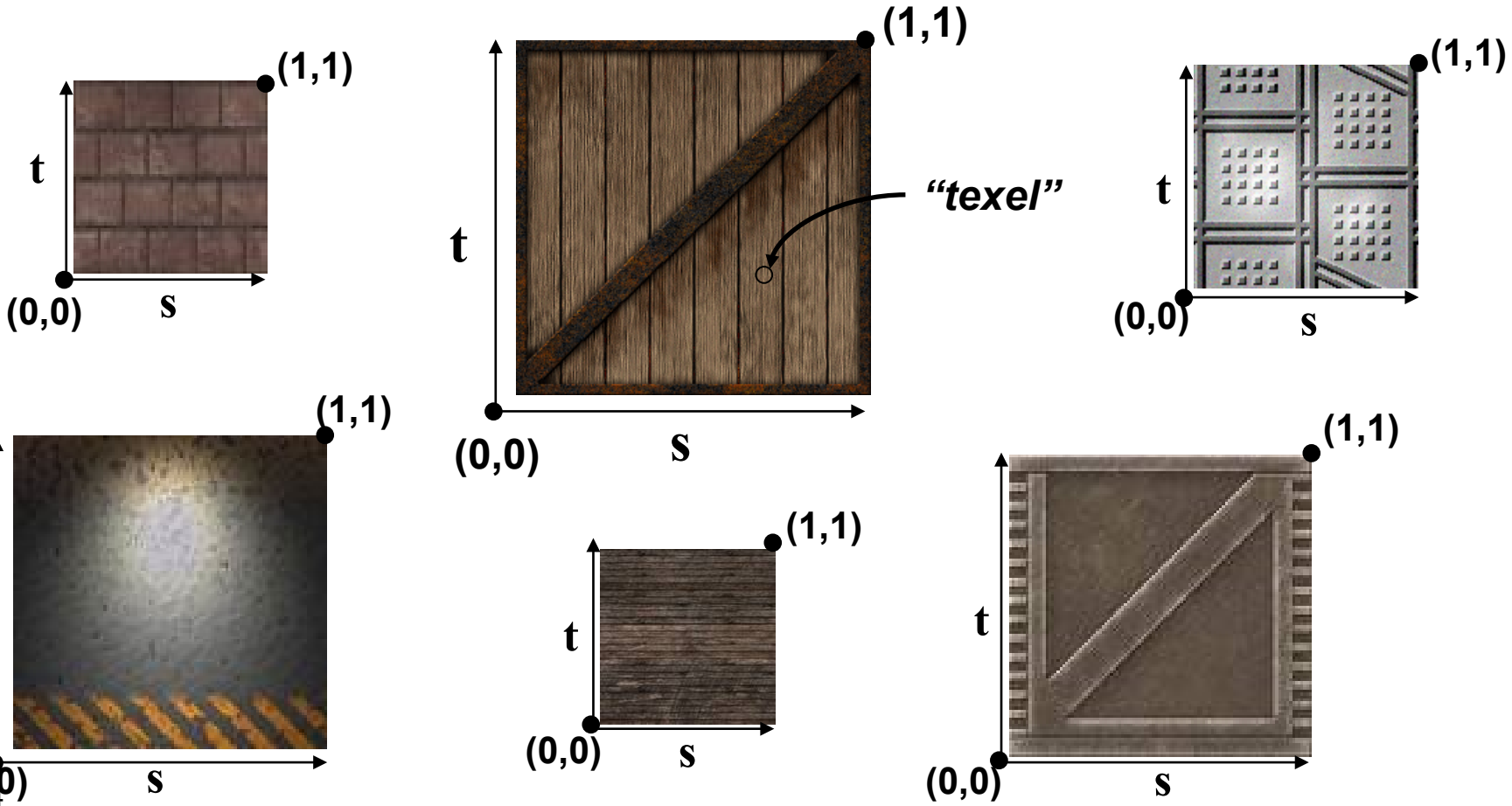


Image Object Polygons “Texture-mapped” object

Texture Space

Textures have their own *coordinate space*:

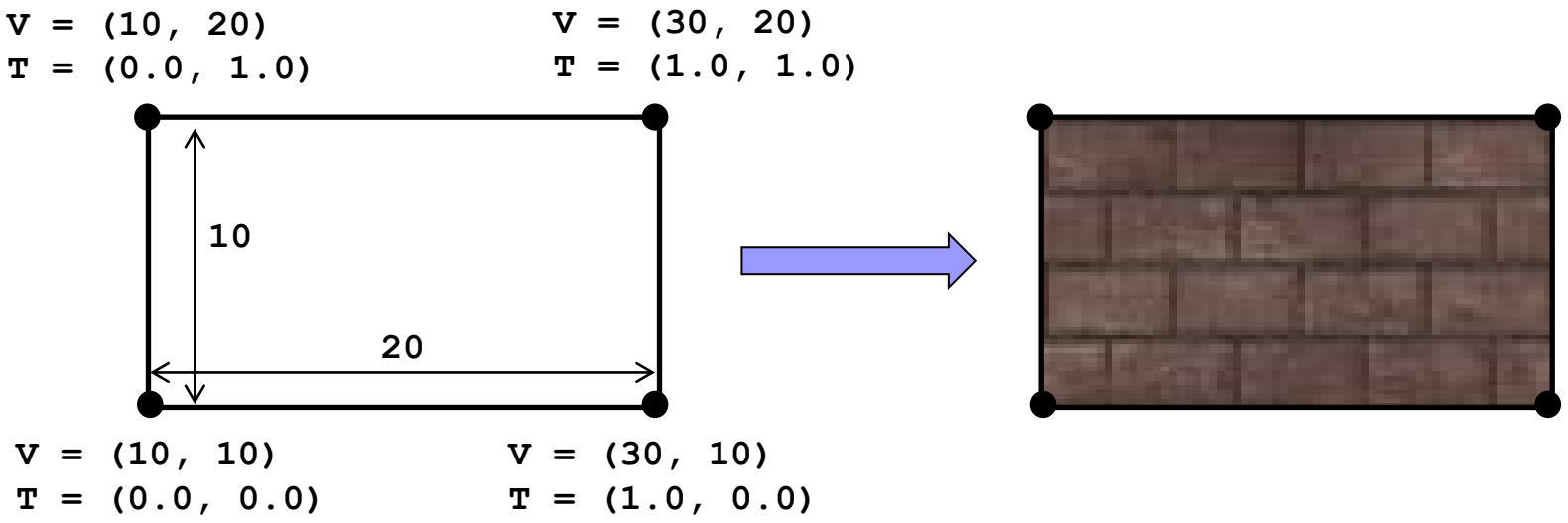


4

Vertex Texture Coordinates

Each *vertex* has an associated *texture coordinate*

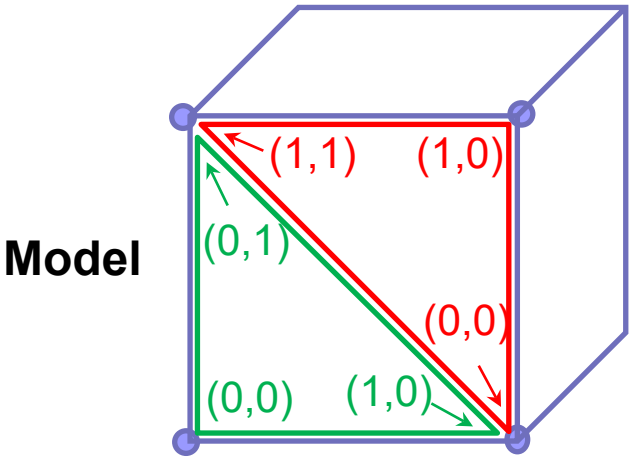
- Texture coordinates can be set by the program

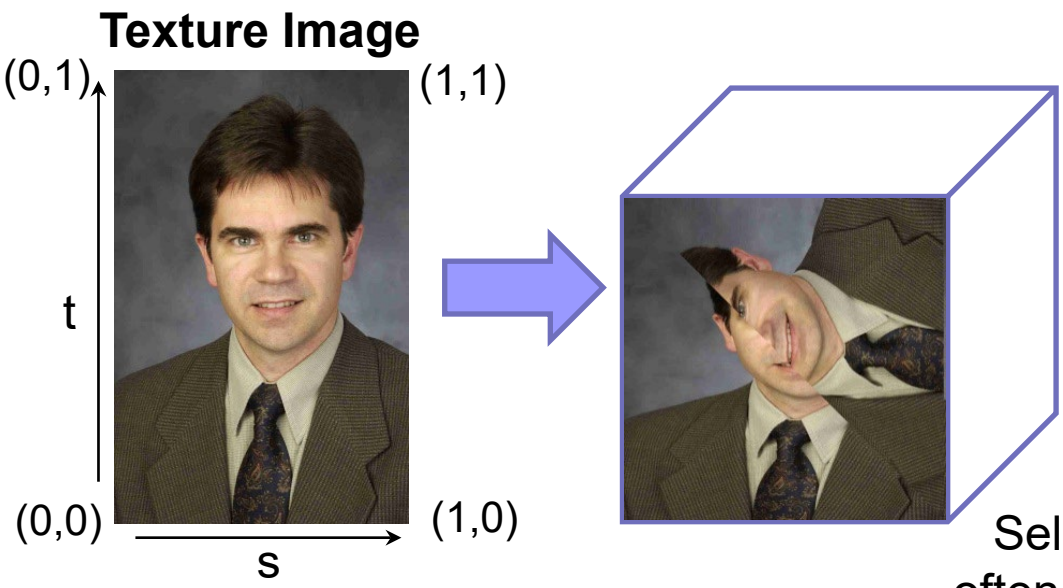


Example

Texture coordinates typically range from (0,0) to (1,1)

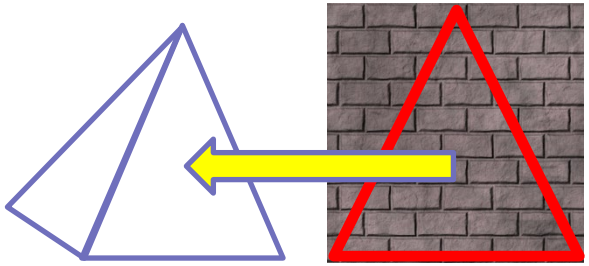
axes typically labeled s, t

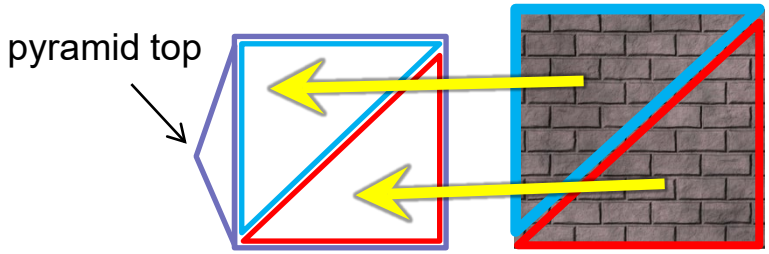




Selected pixels often called “texels”

Constructing texture coordinates for a pyramid






```
vertices          texture coordinates
(-1.0,-1.0, 1.0)      (0, 0)        // front face
( 1.0,-1.0, 1.0)      (1, 0)
( 0.0, 1.0, 0.0)     (.5, 1)

( 1.0,-1.0, 1.0)      (0, 0)        // right face
( 1.0,-1.0,-1.0)      (1, 0)
( 0.0, 1.0, 0.0)     (.5, 1)

( 1.0,-1.0,-1.0)      (0, 0)        // back face
(-1.0,-1.0,-1.0)      (1, 0)
( 0.0, 1.0, 0.0)     (.5, 1)
etc.
```

combining light and textures

Color = textureColor * (ambientLight + diffuseLight) + specularLight

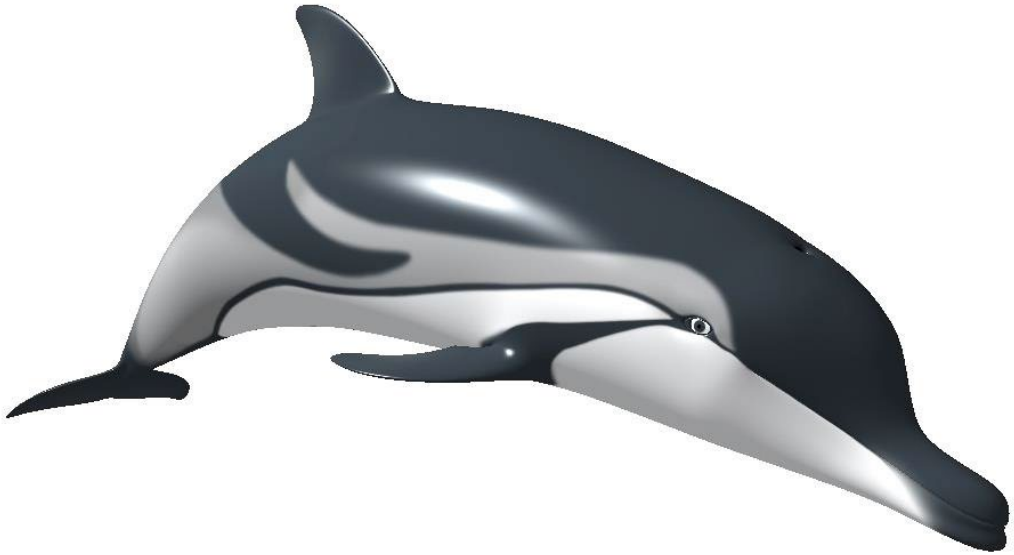
or

Color = textureColor * (ambientLight + diffuseLight + specularLight)

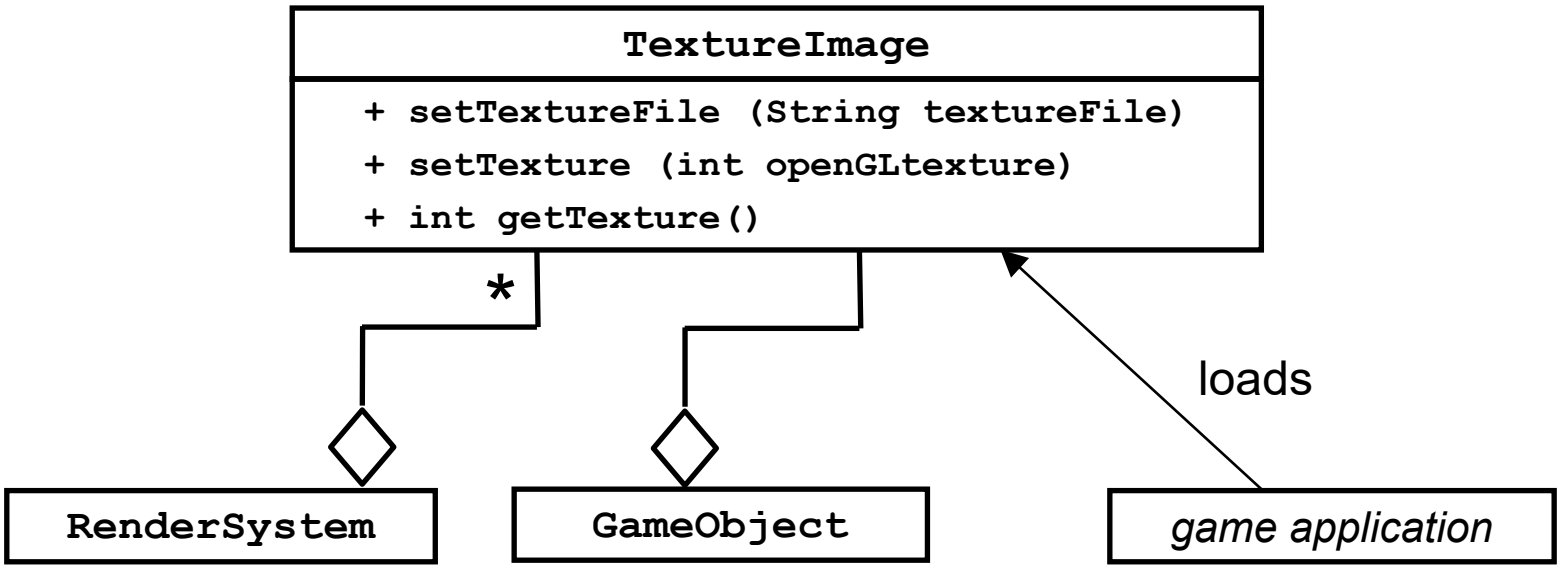
or

Color = (ambLight * ambMaterial) + (diffLight * diffMaterial) + specLight

fragColor = 0.5 * textureColor + 0.5 * lightColor



TAGE Texture classes



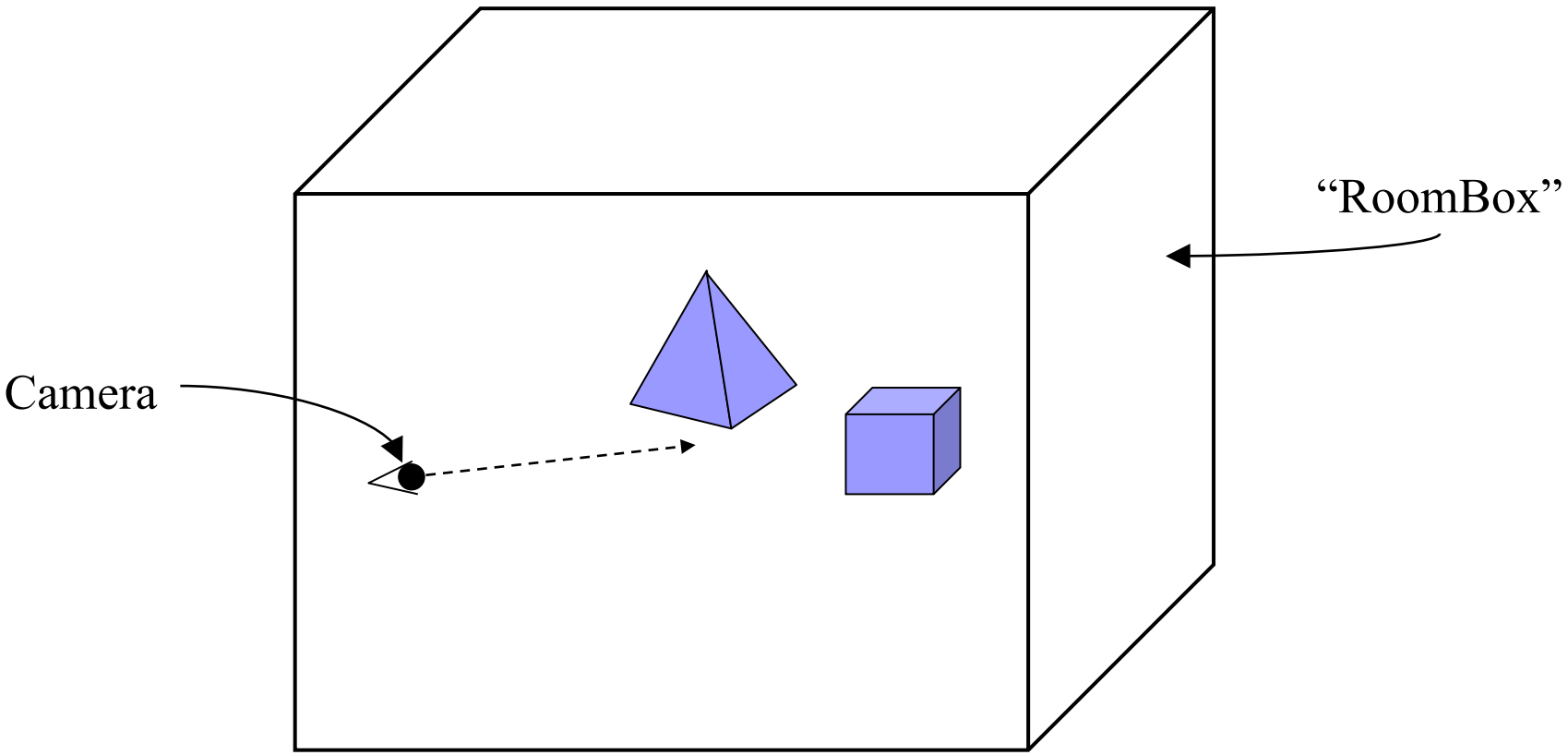

```
@Override
public void loadTextures()
{   earth = new TextureImage("earth.jpg");
    brick = new TextureImage("brick1.jpg");
    doltx = new TextureImage("Dolphin_HighPolyUV.png");
}
...
dolphin = new GameObject(GameObject.root(), dolS, doltx);
```

Game World Background

- Real scenes always have “background”
- Game world background MUST be 3D

  Why?
- Indoors: room walls
- Outdoors: horizon scenery

RoomBoxes



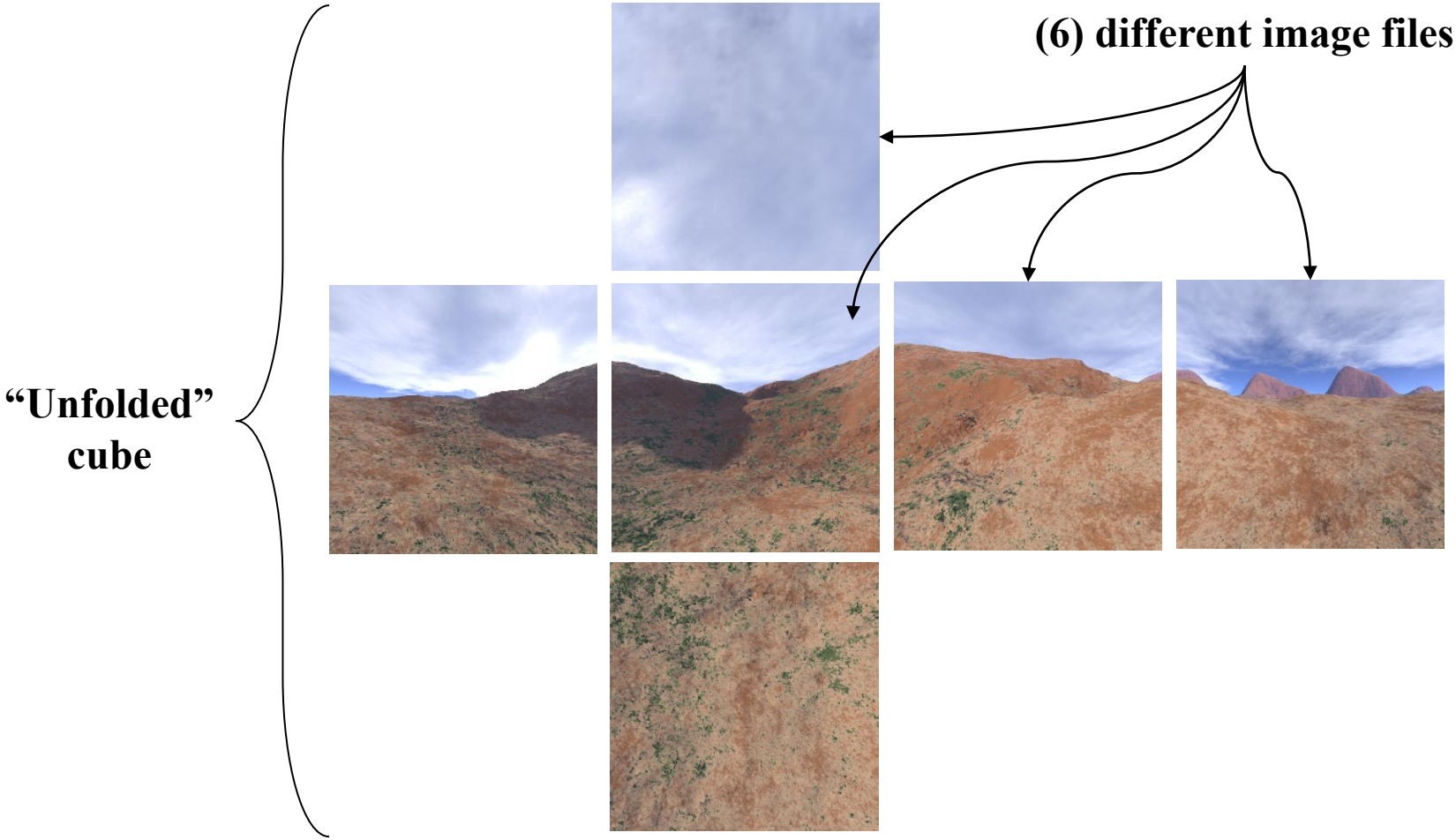
SkyBoxes

But what about *outdoor games ??*

Solution: "SkyBox"

- Texture-mapped *outdoor scene*
- Can be mapped onto different geometries
  - Rectangles
  - Cube
  - Hemisphere
  - …

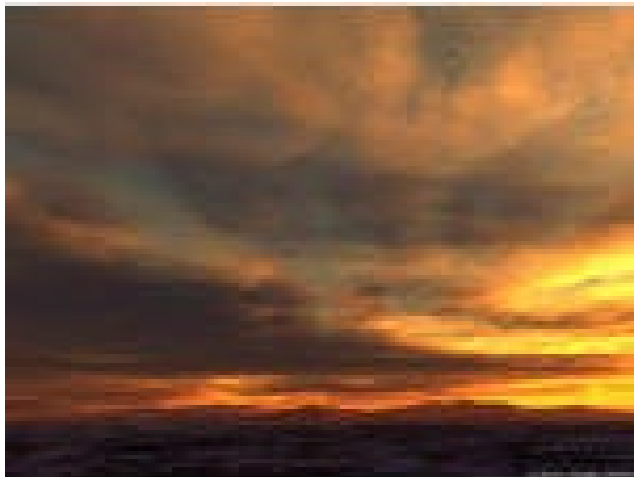
Texture Cube Maps

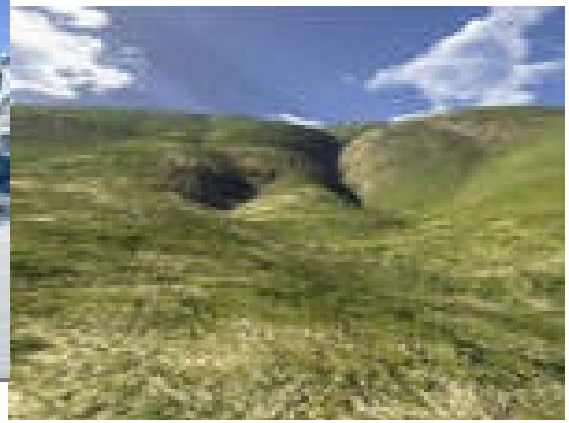


Creating Texture Cube Maps

- Create a 3D scene
- Place camera in middle with 90° FOV
- Render images in each of six directions
- Some tools:
  - **Terragen**
  - **Blender**
  - **Bryce**
  - **SkyPaint**
  - **3DStudio Max**
  - **Maya**

Terragen Example Scenes



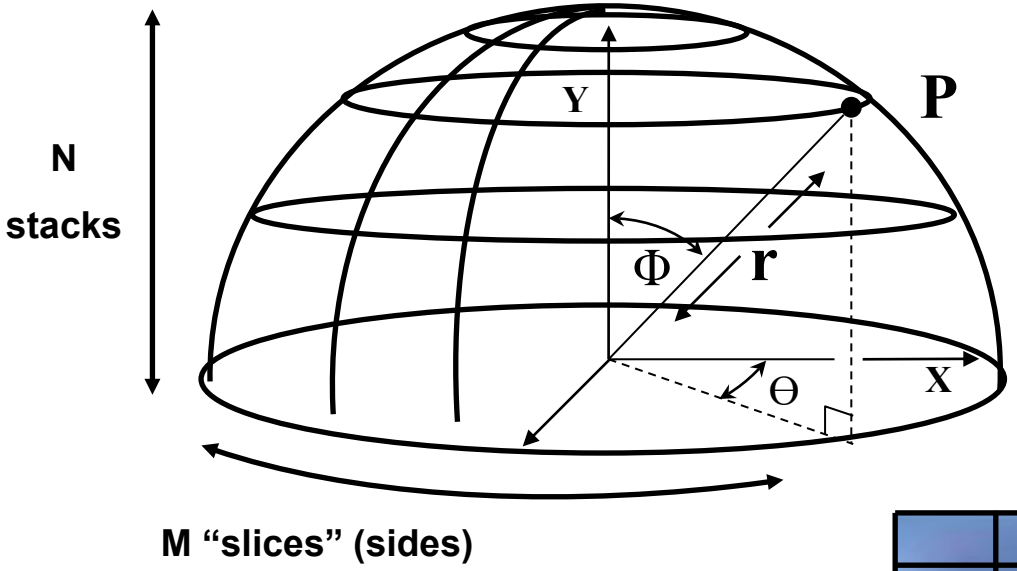


download Terragen: http://www.planetside.co.uk

SkyBoxes challenges:

- Requires six textures
  (can be one image or six separate images)
- "Cube" can cause distortion near corners
- Can show artifacts at texture seams
  mismatches in adjacent texture's pixels
- Inconsistent definitions of "front" & "back"

SKYDOMES



Cartesian (x,y,z) = Polar (r,θ,Φ)

x = r * cos(θ) * cos(Φ)

y = r * sin(Φ)

z = r * sin(θ) * cos(Φ)

Blender, Photoshop, and many others have tools for manipulating "fish-eye" panoramic photos…

World Box Bounds

- SkyBox should always be “far away”

  no matter where user moves

- Trick: *move* box with camera

  camera always stays at center of box.

  box moves, but does not turn, with camera.

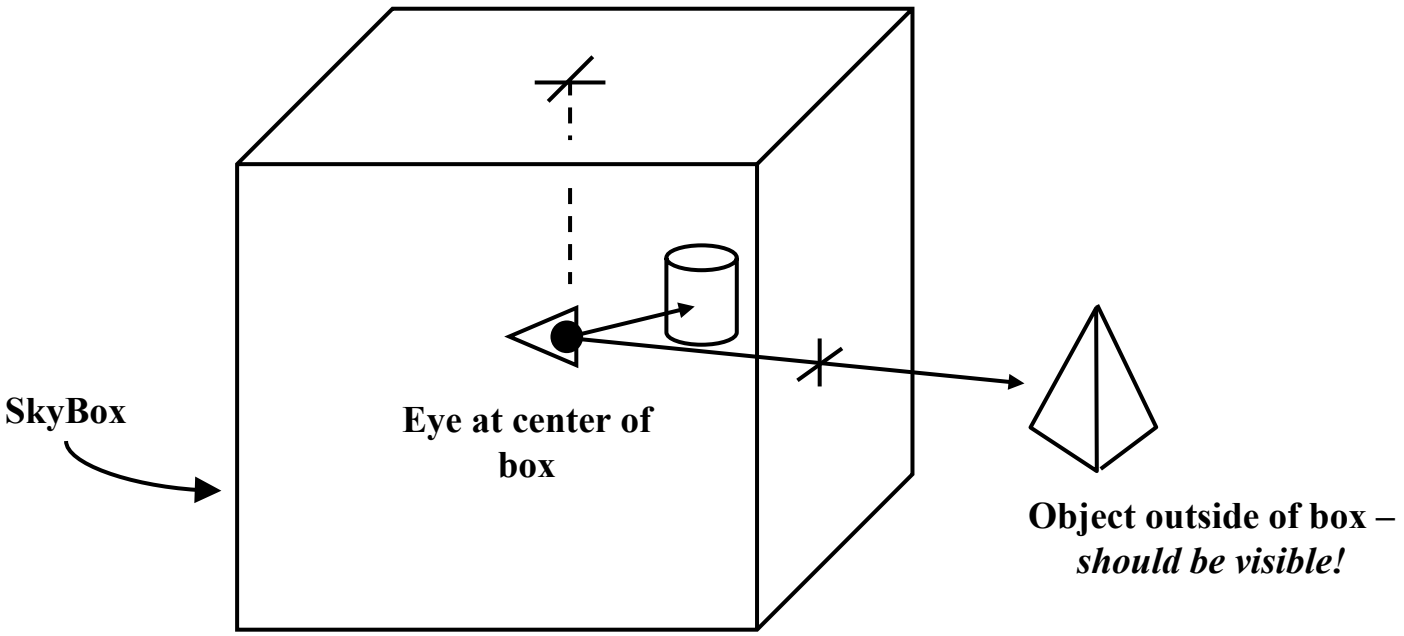
- Most common approach:

  translate box to camera location before drawing

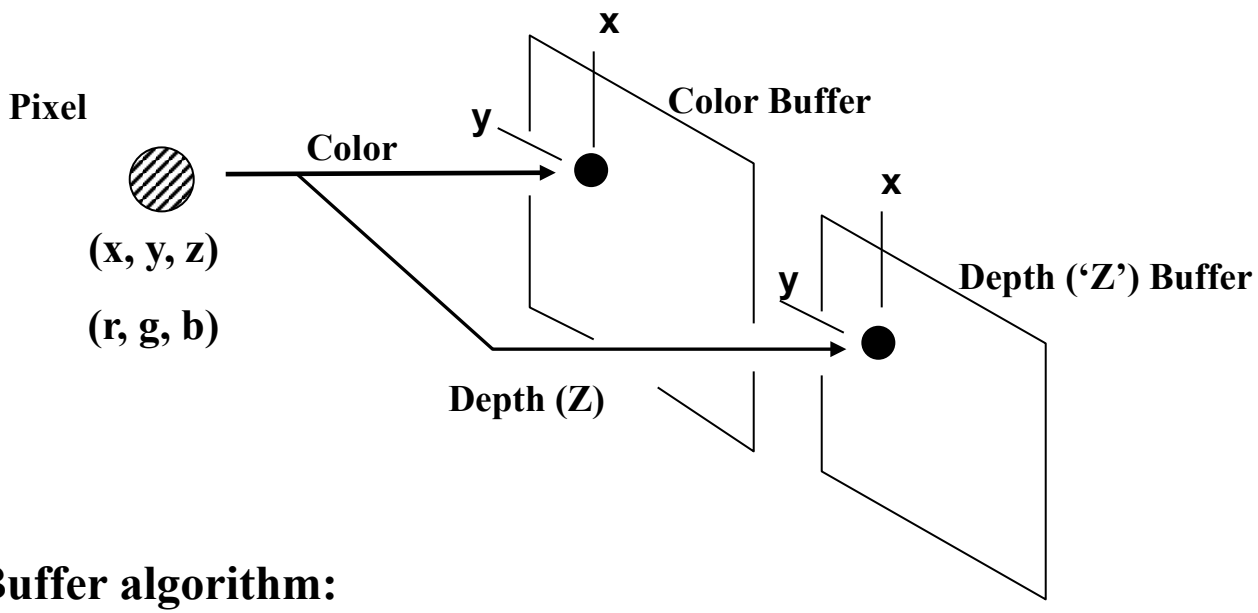
SkyBox Visibility

Problem: objects may lie outside box

HSR means the box will *hide* those objects



The Z-Buffer ("Depth Buffer")



The Z-Buffer algorithm:

```
if (pixel.z < depthBuffer[x,y])
{   colorBuffer[x,y] = pixel(r,g,b);
    depthBuffer[x,y] = pixel(z);
}
```

SkyBox Visibility (continued)

Rendering trick:

- Reset (clear) depth buffer to “max depth”
- Disable depth testing/updating
- Draw SkyBox first
- Re-enable depth testing

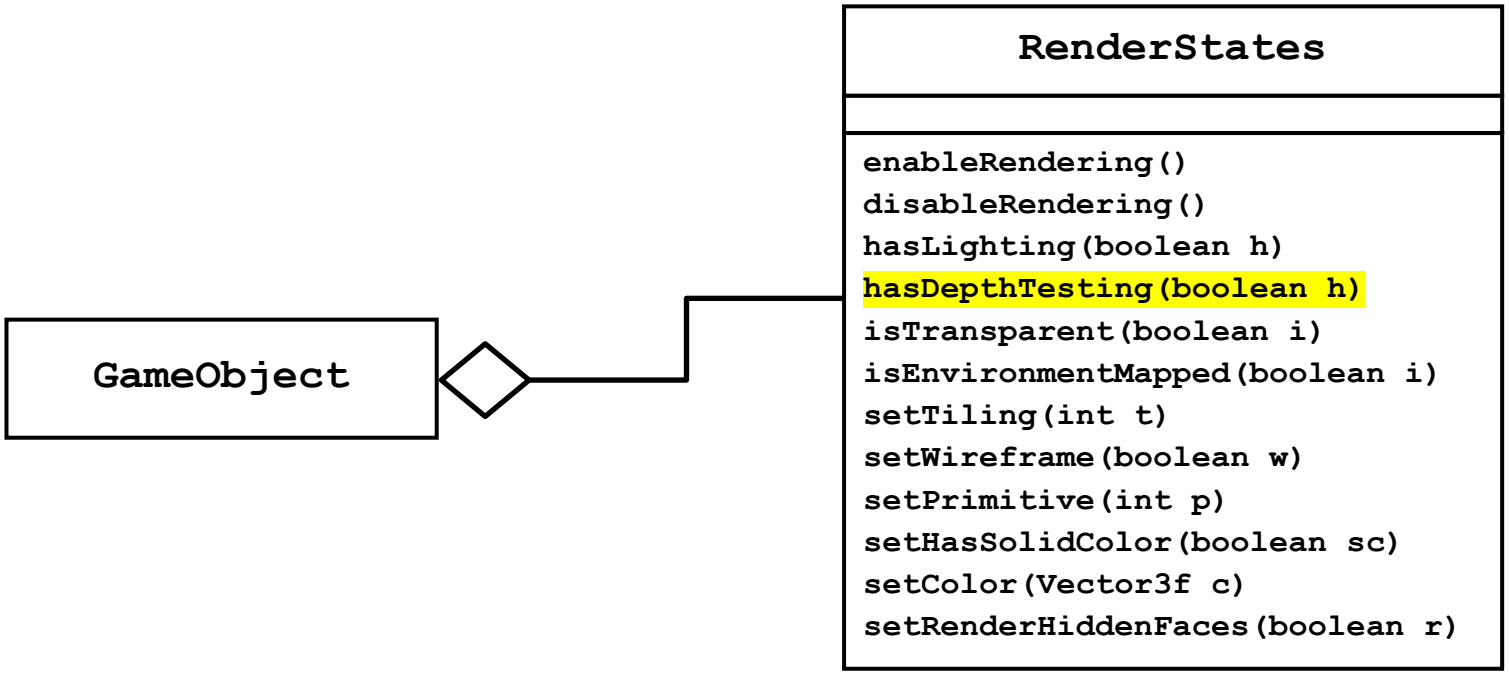
Effect:

- SkyBox pixels will have “maximum depth”
- Subsequent objects drawn with updating enabled will appear “closer”

SkyBox algorithm - summary

```
createSkybox:
{  instantiate a Cube as a game object
   texture the cube with SkyBox textures
      (requires appropriate texture coordinates)
   position the cube at the camera location
}


Update:
{  ...
   get camera location
   translate SkyBox's SceneNode to camera location
      (note - do NOT rotate the skybox cube)
}

Render Frame:
{  clear depth buffer values to "max depth"
   disable depth testing
   draw SkyBox Cube
   enable depth testing
   ...
      (draw the rest of the scene)
}
```

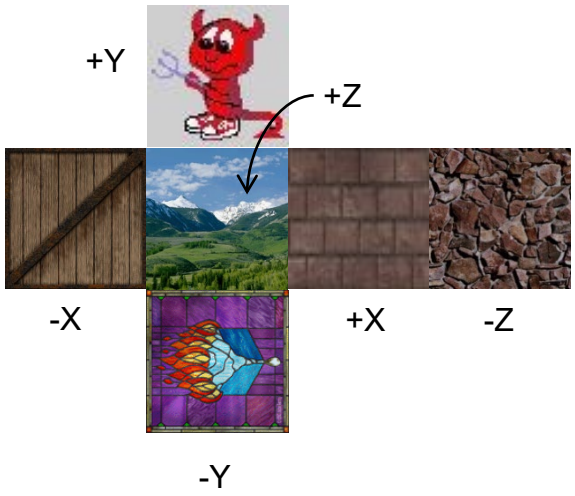
Enabling/Disabling Depth Testing is a *Render State*

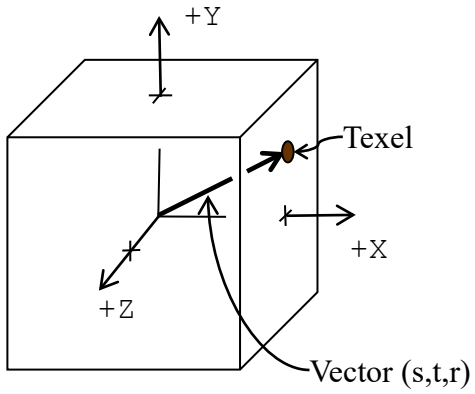


- in some engines, render states are associated with tree scenenodes (e.g., SAGE, JMonkey).
  - ✓ *in this case, render states propagate hierarchically*
- in some engines, render states are associated with renderables, *not* tree scenenodes. This is how TAGE (and RAGE) works.
  - ✓ *In this case, if a render state is intended for an entire subtree (e.g., transparency), the application must set the render state for each GameObject individually.*

OpenGL support for SkyBoxes

- OpenGL cubemap =
  a *single* texture object with *six* 2D faces
- Texture coords (s,t,r) = vector from the *cube center*



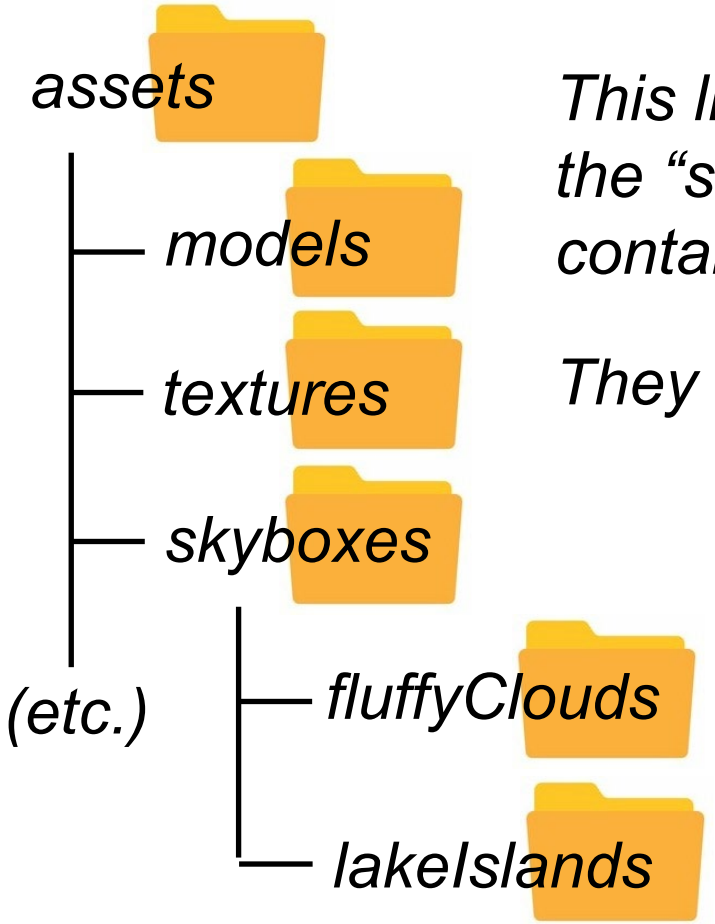


Creating an OpenGL Skybox in TAGE

```
private int fluffyClouds, lakeIslands;
...
// In TAGE, many skyboxes can be loaded.
// But at any given time, at most one skybox is the active skybox.
// TAGE handles building the skybox object and positioning at the camera.
// TAGE also disables depth testing, and renders it before other objects.

public void loadSkyBoxes()
{   fluffyClouds = (engine.getSceneGraph()).loadCubeMap("fluffyClouds");
    lakeIslands = (engine.getSceneGraph()).loadCubeMap("lakeIslands");
    (engine.getSceneGraph()).setActiveSkyBoxTexture(fluffyClouds);
    (engine.getSceneGraph()).setSkyBoxEnabled(true);
}
...

// The active skybox can be changed at any time:
(engine.getSceneGraph()).setActiveSkyBoxTexture(lakeIslands);
(engine.getSceneGraph()).setSkyBoxEnabled(true);

// Or skybox can be disabled, causing no skybox to be rendered:
(engine.getSceneGraph()).setSkyBoxEnabled(false);
```

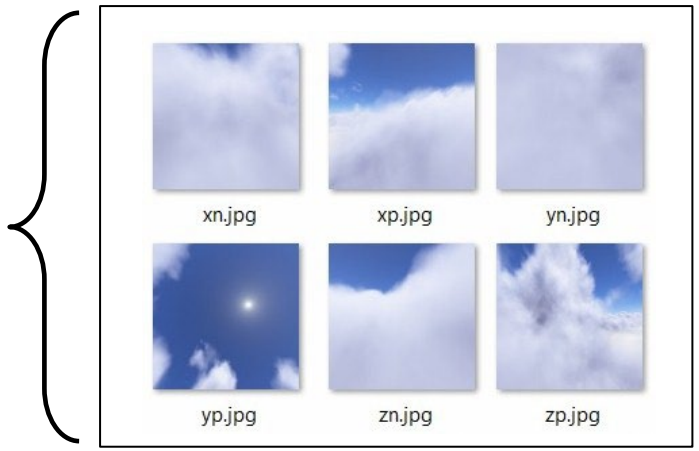
Creating an OpenGL Skybox in TAGE

```
fluffyClouds = (engine.getSceneGraph()).loadCubeMap("fluffyClouds");
```



This line specifies a folder name inside of the "skyboxes" folder in "assets", that contains the six texture files.

They must be named: xp,xn, yp,yn, zp,zn

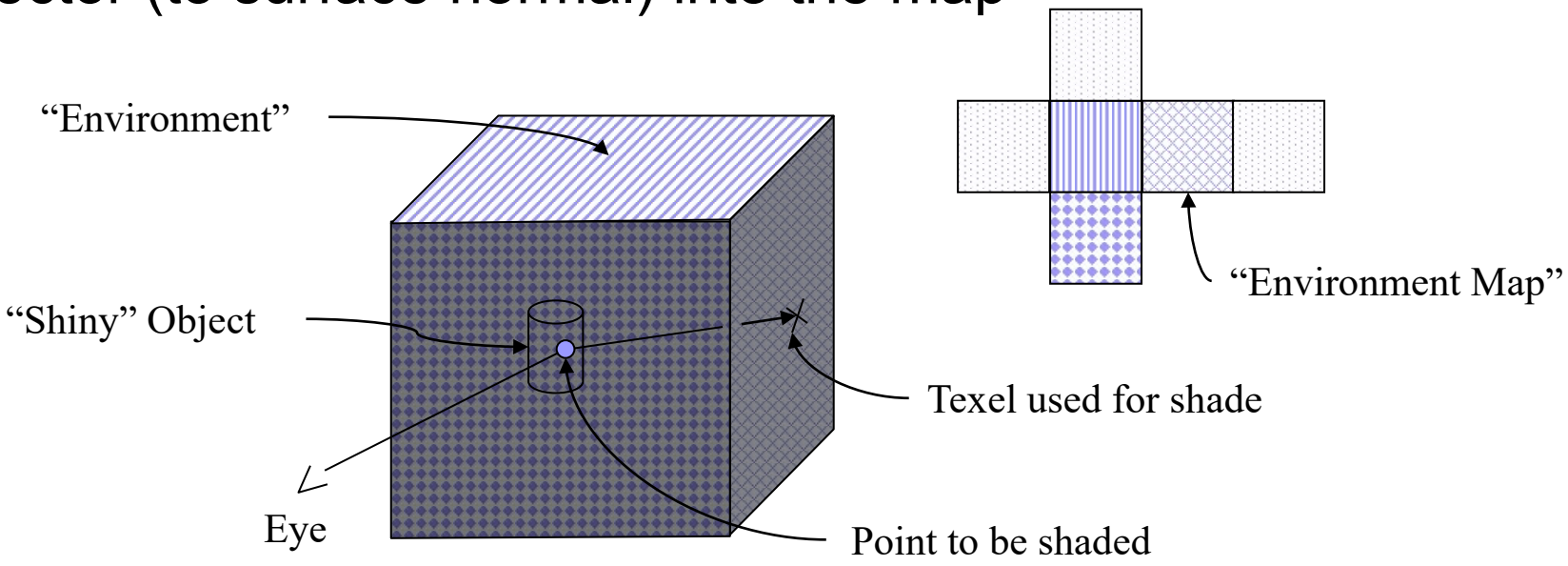


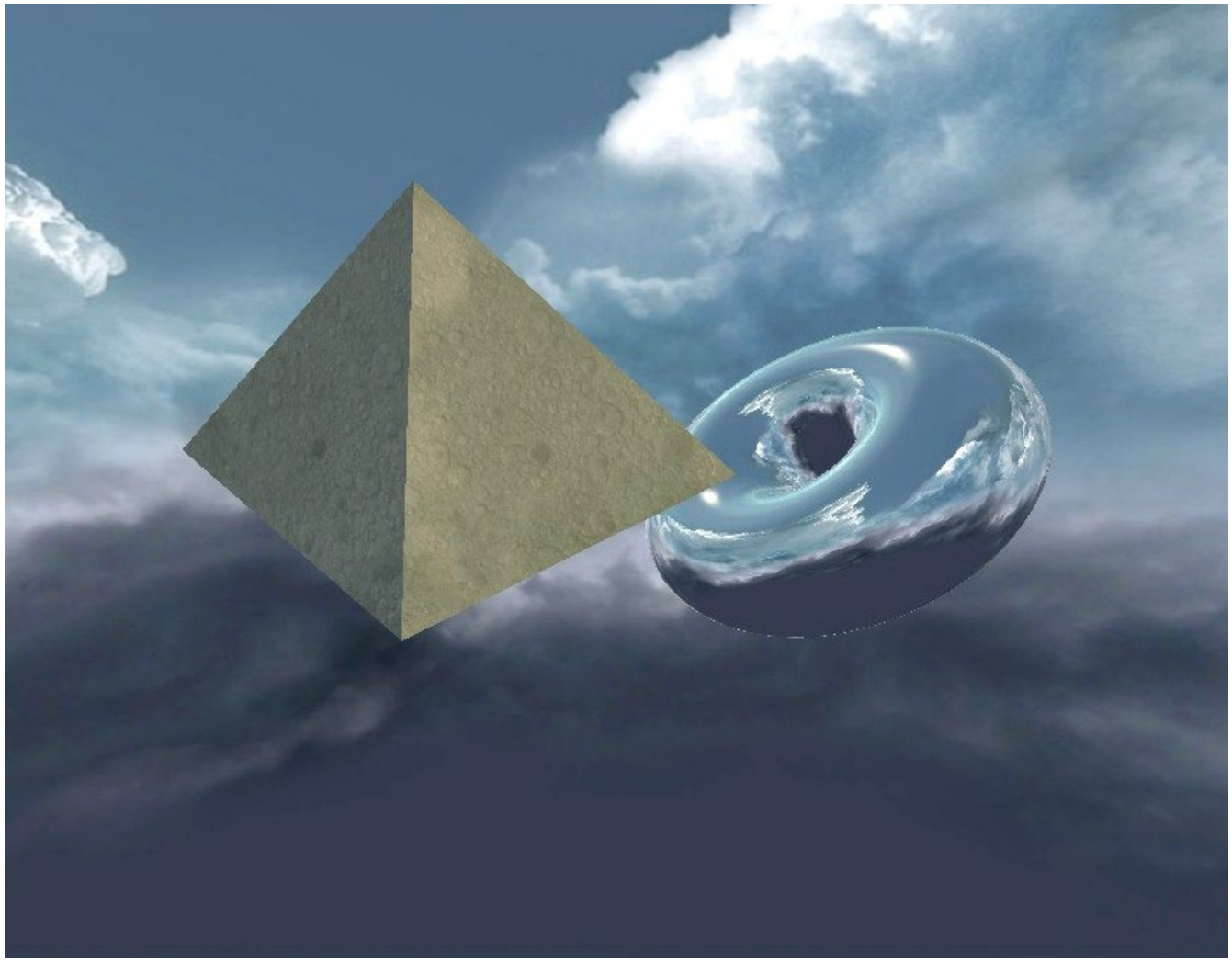
Building a one-image RoomBox in TAGE

```
// In TAGE, a RoomBox object utilizes a single-image texture.
// A RoomBox object doesn't use an OpenGL cube map, so it
// needs to be managed by the programmer.
public void loadShapes()
{  ...
   rbxS = new RoomBox();
}

public void loadTextures()
{  ...
   rbxTx = new TextureImage("room2.png");
}

public void buildObjects()
{  ...
   roombox = new GameObject(GameObject.root(), rbxS, rbxTx);
   initialTranslation = (new Matrix4f()).translation(0,0,-2.0f);
   roombox.setLocalTranslation(initialTranslation);
   roombox.setLocalScale((new Matrix4f()).scaling(3.0f));
   (roombox.getRenderStates()).hasLighting(true);
   (roombox.getShape()).setWindingOrderCCW(true);
}
```

Environment Mapping

- Useful technique for rendering “chrome” objects
  (but only “reflects” the cube map!)
- Create texture cube map describing the “environment”
- Shade object points by following “reflection” of eye vector (to surface normal) into the map





Available in TAGE as a RenderState!