CSc 165
Computer Game Architecture

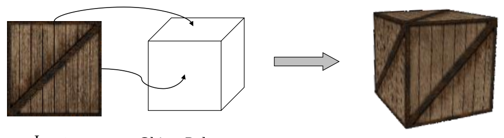
9 - Game World:
textures, skyboxes, etc.

Overview

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

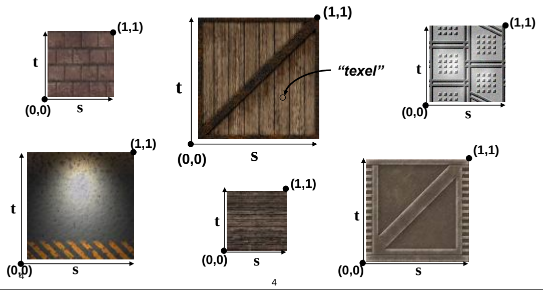
Texture Mapping

Basic idea: attach an "image" to an "object"
- Object == polygon(s)
- Images used this way are called *textures*



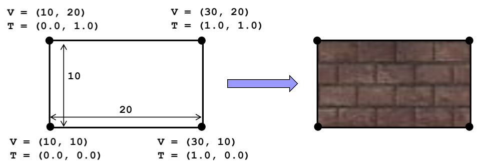
Texture Space

Textures have their own *coordinate space*:

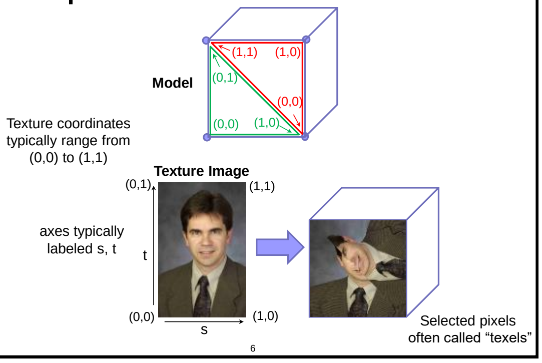


Vertex Texture Coordinates

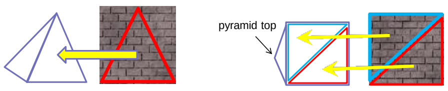
Each *vertex* has an associated *texture coordinate*
- Texture coordinates can be set by the program



Example


Model
(1,1)
(1,0)
(0,1)
(0,0)
(0,0)
(1,0)
Texture coordinates typically range from (0,0) to (1,1)
Texture Image
(0,1)
(1,1)
axes typically labeled s, t
t
(0,0)
s
(1,0)
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)
```

combining light and textures

Color = textureColor * (ambientLight + diffuseLight) + specularLight

or

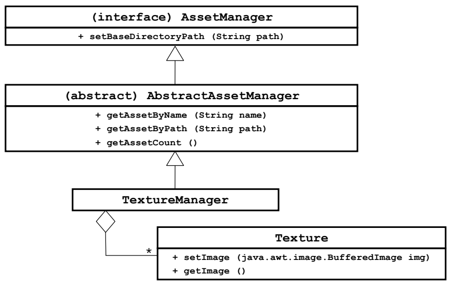
Color = textureColor * (ambientLight + diffuseLight + specularLight)

or

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

fragColor = 0.5 * textureColor + 0.5 * lightColor

RAGE Texture classes


(interface) AssetManager
+ setBaseDirectoryPath (String path)
(abstract) AbstractAssetManager
+ getAssetByName (String name)
+ getAssetByPath (String path)
+ getAssetCount ()
TextureManager
*
Texture
+ setImage (java.awt.image.BufferedImage img)
+ getImage ()

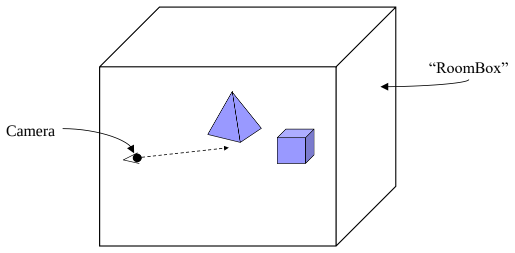
```
TextureManager tm = eng.getTextureManager();
...
Texture redTexture = tm.getAssetByPath("redDolphin.jpg");
```

Game World Background

- Real scenes always have “background”
- Game world background MUST be <u>3D</u>

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

RoomBoxes



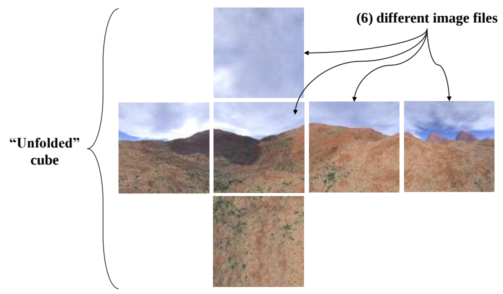
SkyBoxes

But what about *outdoor games* ??

Solution: “<u>SkyBox</u>”

- 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`

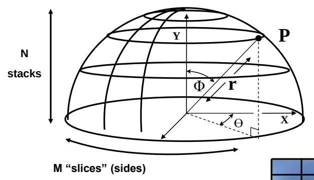
Terragen Example Scenes

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

SkyBoxes challenges:

- Requires <u>six</u> textures
  uses valuable texture memory
- Time-consuming to build
- "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: *<u>move</u>* box with camera
  camera always stays at center of box.
  box moves, but does <u>not</u> turn, with camera.
- Most common approach:
  translate box to camera location before drawing

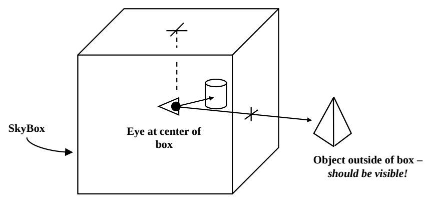
Building a simple SkyBox from scratch

```
// Called from setupScene()
// - creates a simple SkyBox out of a Cube
createSkybox:
{  instantiate a Cube
   instantiate a SceneNode for the Cube, with root as parent
   attach the Cube to the SceneNode
   texture the cube with SkyBox textures
      (requires appropriate texture coordinates)
   position the cube at the camera location
}

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

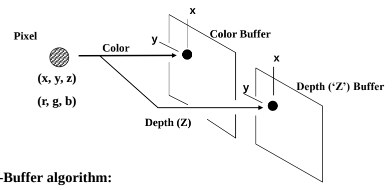
SkyBox Visibility

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:

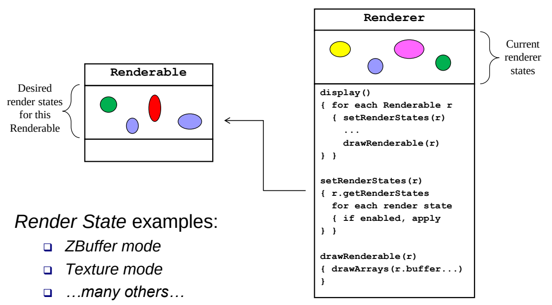
- Reset (clear) depth buffer to "max depth"
- Disable depth testing/updating
- Draw SkyBox first
- Re-enable depth testing

Effect:

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

How can we make this API-independent?

Render States

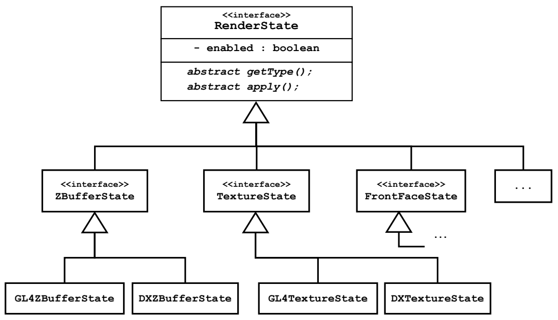


```
display()
{ for each Renderable r
  { setRenderStates(r)
    ...
    drawRenderable(r)
} }

setRenderStates(r)
{ r.getRenderStates
  for each render state
  { if enabled, apply
} }

drawRenderable(r)
{ drawArrays(r.buffer...)
}
```

Render State examples:

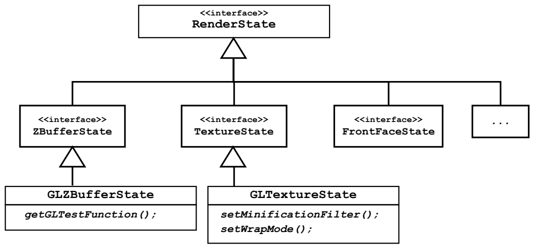
- *ZBuffer mode*
- *Texture mode*
- *…many others…*

Render States (cont.)



- in some engines, render states are associated with scene nodes (e.g., SAGE, JMonkey).
  - ✓ *in this case, render states propagate hierarchically*
- in some engines, render states are associated with entities or renderables, *not* scene nodes. This is how 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 node individually.*

- *Different render state types can have different capabilities and functionality*



Creating a Skybox in RAGE

```
void setupScene()
{     . . .
      Configuration conf = eng.getConfiguration();
      TextureManager textureMgr = getEngine().getTextureManager();

      textureMgr.setBaseDirectoryPath(conf.valueOf("assets.skyboxes.path"));
      Texture front = textureMgr.getAssetByPath("front.jpeg");
      Texture back = textureMgr.getAssetByPath("back.jpeg");
      Texture left = textureMgr.getAssetByPath("left.jpeg");
      Texture right = textureMgr.getAssetByPath("right.jpeg");
      Texture top = textureMgr.getAssetByPath("top.jpeg");
      Texture bottom = textureMgr.getAssetByPath("bottom.jpeg");
      textureMgr.setBaseDirectoryPath(conf.valueOf("assets.textures.path"));

      // cubemap textures must be flipped up-side-down to face inward;
      // all textures must have the same dimensions,
      //   so any image's height will do
      AffineTransform xform = new AffineTransform();
      xform.translate(0, front.getImage().getHeight());
      xform.scale(1d, -1d);
```

continued…

```
      front.transform(xform);
      back.transform(xform);
      left.transform(xform);
      right.transform(xform);
      top.transform(xform);
      bottom.transform(xform);

      SkyBox sb = sm.createSkyBox("mySkyBox");
      sb.setTexture(front, SkyBox.Face.FRONT);
      sb.setTexture(back, SkyBox.Face.BACK);
      sb.setTexture(left, SkyBox.Face.LEFT);
      sb.setTexture(right, SkyBox.Face.RIGHT);
      sb.setTexture(top, SkyBox.Face.TOP);
      sb.setTexture(bottom, SkyBox.Face.BOTTOM);
      sm.setActiveSkyBox(sb);
}
```