Texturing

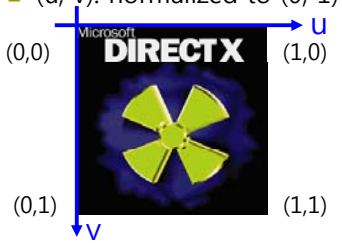
305890
Spring 2013
4/26/2013
Kyoung Shin Park

Overview

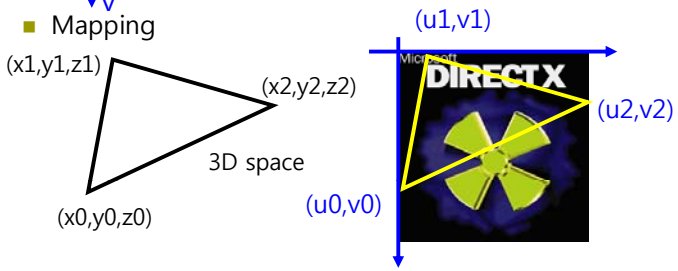
- Texture coordinates
- Create and enable textures
- Texture filters
- Mipmaps
- Address Modes
- Using a Basic Effect with Texturing
- Creating a Custom Effect with Texturing

Texture Coordinates

- Texture Coordinates
  - (u, v): normalized to (0, 1)

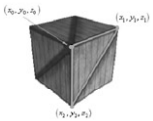


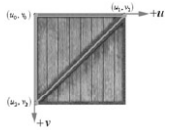
  - Mapping



Texture Coordinates

- Vertex structure include **texture coordinates**


3D Triangle


Corresponding Texture Triangle

```
VertexPositionNormalTexture[] vertices;

vertices[0].Position = LowerLeft;
vertices[0].TextureCoordinate = textureLowerLeft;   // (0, 1)
vertices[1].Position = UpperLeft;
vertices[1].TextureCoordinate = textureUpperLeft;   // (0, 0)
vertices[2].Position = LowerRight;
vertices[2].TextureCoordinate = textureLowerRight;  // (1, 1)
vertices[3].Position = UpperRight;
vertices[3].TextureCoordinate = textureUpperRight;  // (1, 0)
```

Creating and Enabling Textures

- Load an image file into **Texture2D** object.
  - To create a texture from an image "crate.jpg"
  - Texture2D texture = Content.Load<Texture2D>("crate");
  - This function can load any of the following image formats: BMP, DDS, DIB, HDR, JPG, PFM, PNG, PPM, and TGA.

Creating and Enabling Textures

- To use different textures but are drawn using the same effect:

```
// using BasicEffect
foreach (EffectPass pass in effect.CurrentTechnique.Passes) {
    pass.Apply();
    effect.Texture = mTex0;
    drawQuadUsingTex();

    pass.Apply();
    effect.Texture = mTex1;
    drawQuadUsingTex();
    pass.End();
}
```

Creating and Enabling Textures

- To set a created Texture2D object to texture effect parameter, call effect.Parameters["key"].SetValue("value").

```
// TextureEffect.fx file
texture DiffuseTexture;
sampler2D DiffuseSampler = sampler_state {
    Texture = <DiffuseTexture>;
};

// texture mapping program
Texture2D texture = Content.Load<Texture2D>("xna_logo");
effect.Parameters["DiffuseTexture"].SetValue(texture);
drawQuadUsingTex();
```

Filters

- When the texture triangle is smaller than the screen triangle, the texture triangle is magnified to fit.
- When the texture triangle is large than the screen triangle, the texture triangle is minified to fit.
- Mapping filter
  - MAGFILTER
  - MINFILTER

```
texture DiffuseTexture;
sampler DiffuseSampler = sampler_state          {
      Texture = < DiffuseTexture >;
      MinFilter = LINEAR;
      MagFilter = LINEAR;
};
```

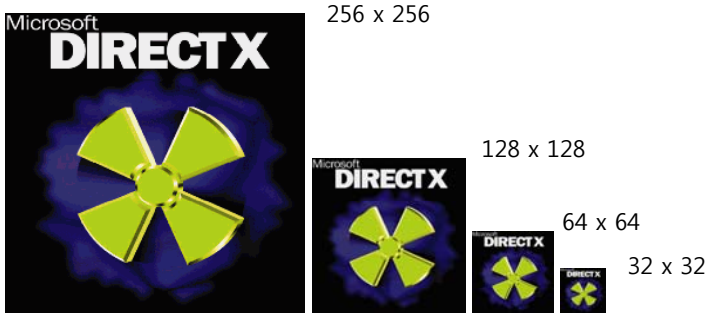
Filters

- 3 types of filters
  - Nearest point sampling: POINT, poor quality, faster (default)
  - Linear filtering: LINEAR, high quality, relatively fast (recommended)
  - Anisotropic filtering: Anisotropic, higher quality, relatively slow.
    - Must also set D3DSAMP_MAXANISOTROPY level (to determine the quality of the anisotropic filtering). (Default is 1)

```
texture DiffuseTexture;
sampler DiffuseSampler2 = sampler_state {
    Texture = < DiffuseTexture >;
    MinFilter = Anisotropic;
    MagFilter = LINEAR;
    MaxAnisotropy = 4;
};
```

Mipmaps

- We can create a chain of mipmaps for a texture.
  - The idea is to take a texture and create a series of smaller, lower-resolution textures, but customizing the filtering for each of these levels.



Mipmap

- Mipmap filter:
  - NONE: disable mipmapping
  - POINT: chooses the mipmap level that is closest in size to the screen triangle. Once that level is chosen, XNA filters that level based on the specified min and mag filters.
  - LINEAR: takes the two mipmap levels that are closest in size to the screen triangle, filters each level with the min and mag filters, and finally linearly combines these two levels to form the final color values.

```
MipFilter = Filter;
```

Address Modes

- The texture coordinates that go outside [0, 1] range is defined by XNA address mode:
  - Address mode:
    - Wrap
    - Border color (not supported in XNA 4.0)
    - Clamp
    - mirror
  - TextureAddressMode enum type
    - WRAP: repeat the texture on every integer junction
    - MIRROR: every other row and column is a mirrors image of the preceding row or column
    - CLAMP: smear the color of edge pixels
    - BORDER: use the border color, for any texture coordinates outside the range

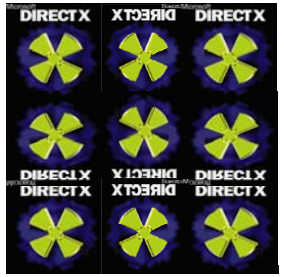
Address Modes

```
// wrap
sampler TexS = sampler_state {
    Texture = <gTex>;
    MinFilter = LINEAR; MagFilter = LINEAR; MipFilter = LINEAR;
    AddressU = WRAP; AddressV = WRAP; };
// border color
sampler TexS = sampler_state {
    Texture = <gTex>;
    MinFilter = LINEAR;     MagFilter = LINEAR; MipFilter = LINEAR;
    AddressU = BORDER; AddressV = BORDER; BorderColor = 0xff0000ff; };
// clamp
sampler TexS = sampler_state {
    Texture = <gTex>;
    MinFilter = LINEAR; MagFilter = LINEAR; MipFilter = LINEAR;
    AddressU = CLAMP; AddressV = CLAMP; };
// mirror
sampler TexS = sampler_state {
    Texture = <gTex>;
    MinFilter = LINEAR; MagFilter = LINEAR; MipFilter = LINEAR;
    AddressU = MIRROR; AddressV = MIRROR; };
```

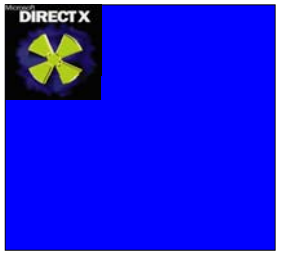
Address Modes

- Quad (0,0),(0,3),(3,0),(3,3):

wrap



mirror

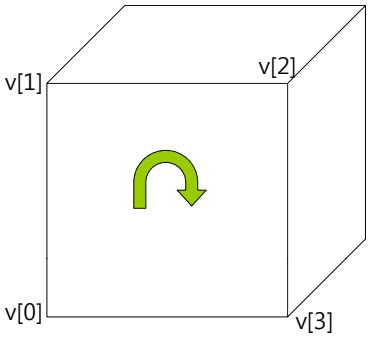


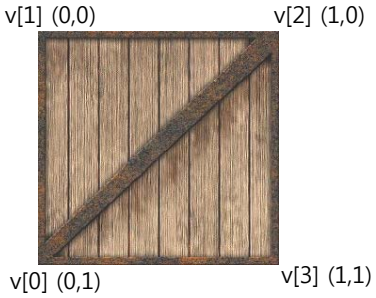
border color

clamp

TextureBox Demo

- Add a texture to a cube.
  1. Specifying the texture coordinates.
  2. Creating the texture using Content.Load<Texture2D>("filename").
  3. Sampling the Texture.
  4. Setting the Effect Texture





TextureBox Demo

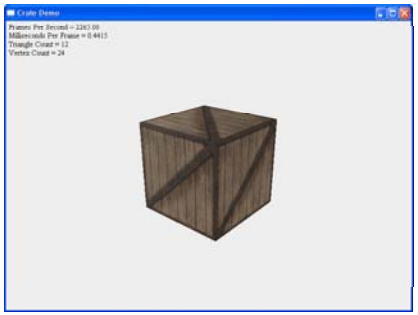
```
// Define the vertices of our box
Vector3 topLeftFront = new Vector3( -1.0f, 1.0f, 1.0f );
Vector3 topRightFront =  new Vector3( 1.0f, 1.0f, 1.0f );
Vector3 bottomRightFront = new Vector3( 1.0f, -1.0f, 1.0f );
Vector3 bottomLeftFront =  new Vector3( -1.0f, -1.0f, 1.0f );
Vector3 topLeftBack =  new Vector3( -1.0f, 1.0f, -1.0f );
Vector3 topRightBack =  new Vector3( 1.0f, 1.0f, -1.0f );
Vector3 bottomRightBack =  new Vector3( 1.0f, -1.0f, -1.0f );
Vector3 bottomLeftBack =  new Vector3( -1.0f, -1.0f, -1.0f );

// Define our normals so we can use directional lighting
Vector3 frontNormal = new Vector3( 0.0f, 0.0f, 1.0f );
Vector3 backNormal = new Vector3( 0.0f, 0.0f, -1.0f );
Vector3 topNormal = new Vector3( 0.0f, 1.0f, 0.0f );
Vector3 bottomNormal = new Vector3( 0.0f, -1.0f, 0.0f );
Vector3 rightNormal = new Vector3( 1.0f, 0.0f, 0.0f );
Vector3 leftNormal = new Vector3( -1.0f, 0.0f, 0.0f );
```

TextureBox Demo

```
// Define our texture coordinates
Vector2 textureBottomLeft = new Vector2( 0.0f, 1.0f );
Vector2 textureTopLeft = new Vector2( 0.0f, 0.0f );
Vector2 textureTopRight = new Vector2( 1.0f, 0.0f );
Vector2 textureBottomRight = new Vector2( 1.0f, 1.0f );
// Front face (CW winding order).
vertices[ 0 ] = new VertexPositionNormalTexture(
            bottomLeftFront, frontNormal, textureBottomLeft );
vertices[ 1 ] = new VertexPositionNormalTexture(
            topLeftFront, frontNormal, textureTopLeft );
vertices[ 2 ] = new VertexPositionNormalTexture(
            topRightFront, frontNormal, textureTopRight );
vertices[ 3 ] = new VertexPositionNormalTexture(
            bottomLeftFront, frontNormal, textureBottomLeft );
vertices[ 4 ] = new VertexPositionNormalTexture(
            topRightFront, frontNormal, textureTopRight );
vertices[ 5 ] = new VertexPositionNormalTexture(
            bottomRightFront, frontNormal, textureBottomRight );
```

TextureBox Demo

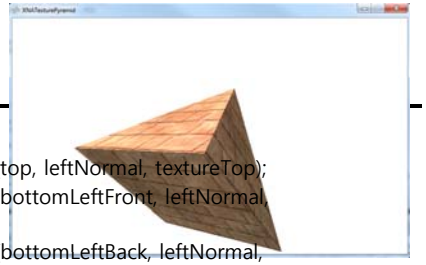
```
sampler TexS = sampler_state
{
    Texture = <gTex>;
    MinFilter = Anisotropic;
    MagFilter = LINEAR;
    MipFilter = LINEAR;
    MaxAnisotropy = 8;
    AddressU = WRAP;
    AddressV = WRAP;
};
```



TexturePyramid Demo

```
// Define the vertices of our pyramid
Vector3 top = new Vector3( 0.0f, 1.0f, 0.0f );
Vector3 bottomRightFront = new Vector3( 1.0f, -1.0f, 1.0f );
Vector3 bottomLeftFront =  new Vector3( -1.0f, -1.0f, 1.0f );
Vector3 bottomRightBack =  new Vector3( 1.0f, -1.0f, -1.0f );
Vector3 bottomLeftBack =  new Vector3( -1.0f, -1.0f, -1.0f );
// Define our normals for lighting (same as TextureBox) 중간 생략..
// Define our texture coordinates
Vector2 textureTop = new Vector2( 0.5f, 0.0f );
Vector2 textureTopLeft = new Vector2( 0.0f, 0.0f );
Vector2 textureTopRight = new Vector2( 1.0f, 0.0f );
Vector2 textureBottomLeft = new Vector2( 0.0f, 1.0f );
Vector2 textureBottomRight = new Vector2( 1.0f, 1.0f );
// Front face (CW winding order).
vertices[ 0 ] = new VertexPositionNormalTexture(top, frontNormal, textureTop);
vertices[ 1 ] = new VertexPositionNormalTexture(
            bottomRightFront, frontNormal, textureBottomRight);
vertices[ 2 ] = new VertexPositionNormalTexture(
            bottomLeftFront, frontNormal, textureBottomLeft);
```

TexturePyramid Demo

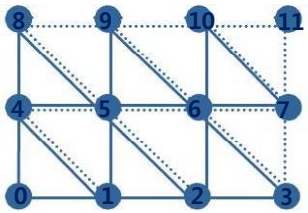


```
// Left face.
Vertices[3] = new VertexPositionNormalTexture(top, leftNormal, textureTop);
Vertices[4] = new VertexPositionNormalTexture(bottomLeftFront, leftNormal,
textureBottomRight);
Vertices[5] = new VertexPositionNormalTexture(bottomLeftBack, leftNormal,
textureBottomLeft);
//  중간생략..
// Bottom face
Vertices[12] = new VertexPositionNormalTexture(bottomRightFront, bottomNormal,
textureTopRight);
Vertices[13] = new VertexPositionNormalTexture(bottomRightBack, bottomNormal,
textureBottomRight);
Vertices[14] = new VertexPositionNormalTexture(bottomLeftFront, bottomNormal,
textureTopLeft);
Vertices[15] = new VertexPositionNormalTexture(bottomLeftFront, bottomNormal,
textureTopLeft);
Vertices[16] = new VertexPositionNormalTexture(bottomRightBack, bottomNormal,
textureBottomRight);
Vertices[17] = new VertexPositionNormalTexture(bottomLeftBack, bottomNormal,
textureBottomLeft);
```

Tiled Ground Demo

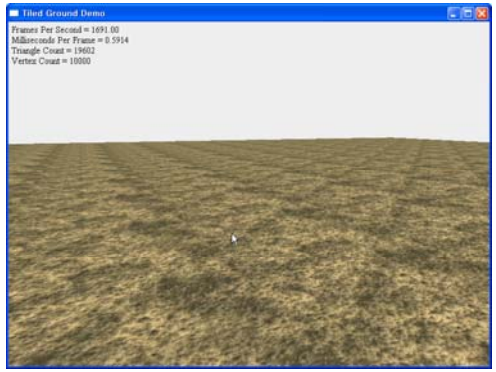
- Tile a ground texture repeatedly over a grid to provide us with a "ground" plane.

```
// tile a texture over a grid mesh
for (int i = 0; i <= numRows; i++) {
    for (int j = 0; j <= numCols; j++) {
        int index = i * (numCols+1) + j;
        v[index].Position = verts[index];                    // position
        v[index].Normal = new Vector3(0.0f, 1.0f, 0.0f);  // normal
        v[index].TextureCoordinate = new Vector2((float)j, (float)i) *
    texScale;// texture
    }
}
```



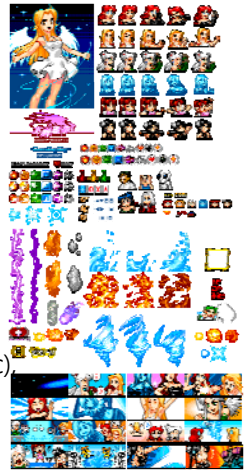
Tiled Ground Demo

```
sampler TexS = sampler_state
{
    Texture = <gTex>;
    MinFilter = Anisotropic;
    MagFilter = LINEAR;
    MipFilter = LINEAR;
    MaxAnisotropy = 8;
    AddressU = WRAP;
    AddressV = WRAP;
};
```



Sprites

- In computer graphics or games, a sprite is a 2D image or animation that is integrated into a larger scene.
- Originally invented as a method of quickly compositing several images together in 2D video games.
- In general, 2D game figures are all referred to as sprites.
- Uses
  - Main character, enemies, any living things, projectiles (rockets, bullets, arrows, rocks, etc), vehicles, etc



Sprite-based graphics

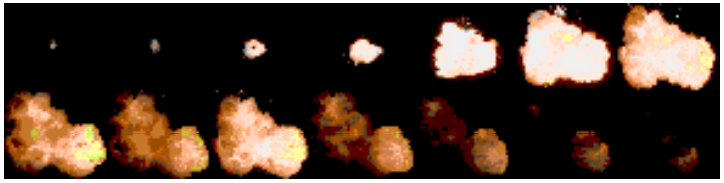
- In the past, the way of doing game graphics
- Still commonly used
  - Mobile phones
  - Internet games
  - Casual games
  - Serious games
- Advantages
  - No graphics hardware required
  - High quality, also on low resolution
  - Much detail
  - Relatively easy to use and control
- Problems
  - Animation speed difficult to control
  - Lot of memory required
  - Fixed viewpoint

Sprite Demo

- Drawing a Sprite.
  1. Define a **SpriteBatch** object as a field on your game class.
  2. Create the SpriteBatch object, passing the current graphics device in LoadContent() method.
     ```
     spriteBatch = new SpriteBatch(GraphicsDevice);
     ```
  3. Load the texture using Content.Load<Texture2D>("filename").
     ```
     texture = Content.Load<Texture2D>("sprite_character");
     ```
  4. In Draw() method,
     ```
     spriteBatch.Begin()
     spriteBatch.Draw(texture, position, Color.White);
     spriteBatch.End();
     ```

Animating Sprites

- "Animating Sprites" represents the different sprites on the screen showing an image with the movement.
- Animation must be:
  - Tied to timer
  - Tied to movement (for main character)



Animating Sprites

- What kinds of data is needed for animating sprites on the screen?
  - Position
  - Z-order
  - Velocity
  - Textures
  - Possible states of sprite – e.g. update position of a bullet
  - Current state of sprite
  - Animation sequences for different states
  - Current frame being displayed (an index)
  - Animation speed

AnimatingSprite Demo

- Drawing a AnimatingSprite.
  - AnimatingSprite demonstrates how to animate a sprite from a texture using a custom class
  - **AnimatedTexture** class loads the m x n tiled image of equal-sized sprite images

```
private AnimatedTexture texture1;
private AnimatedTexture texture2;
private AnimatedTexture texture3;
public Game1() {
...
    texture1 = new AnimatedTexture(Vector2.Zero, 0.0f, 1.0f, 0.5f);
    texture2 = new AnimatedTexture(Vector2.Zero, 30.0f, 3.0f, 0.5f);
    texture3 = new AnimatedTexture(Vector2.Zero, 0.0f, 1.5f, 0.5f);
}
```

AnimatingSprite Demo

```
protected override void LoadContent() {
...
    texture1.Load(Content, "goblintiles", 3, 3, 2); // 3x3 gobline
    texture2.Load(Content, "shipanimated", 4, 2);// 4x1 ship
    texture3.Load(Content, "cavemen", 4, 4, 3);   // 4x4 cavemen
}
protected override void Update(GameTime gameTime) {
...
    float elapsed = (float)gameTime.ElapsedGameTime.TotalSeconds;
    texture1.UpdateFrame(elapsed);
    texture2.UpdateFrame(elapsed);
    texture3.UpdateFrame(elapsed);
}
```

AnimatingSprite Demo

```
protected override void Draw(GameTime gameTime) {
...
    spriteBatch.Begin();
    texture1.DrawFrame(spriteBatch, position1);
    texture2.DrawFrame(spriteBatch, position2);
    texture3.DrawFrame(spriteBatch, position3);
    spriteBatch.End();
}
```