Blending

305890
Spring 2012
5/21/2012
Kyoung Shin Park

Overview

- Blending Equation
- Blend Factors & Mode
- Transparency
- Creating an Alpha Channel Using DX Tex Tool
- Render Semi-Transparent Objects

Blending Equation

- Blending
  - Allows us to blend (combine) the **original pixels** that we are currently rasterizing with the **destination pixels** that were previously rasterized to the back buffer.
  - If P_{ij} is the source pixel we are currently rasterizing, then P_{ij} is blended with the previous ij-th destination pixel on the back buffer b_{ij}.
- Blending Rule
  1. Draw an object with no blending (i.e., Opaque)
  2. Sort blending objects according to the depth from the camera (In viewing space, sort objects by z)
  3. Draw objects in the back to front order.

Blending Equation

- Direct3D Blending Equation

  outputPixel = srcPixel⊗srcBlendFactor + dstPixel⊗dstBlendFactor
  - Each of the above variables is a 4D color vector (r, g, b, a), ⊗ denotes component-wise multiplication
  - outputPixel: the resulting blended pixel
  - srcPixel: the pixel currently being computed that is to be blended with pixel on the back buffer
  - dstPixel: the pixel currently on the back buffer
  - srcBlendFactor & dstBlendFactor: [0,1]

Blending Equation

- Enable/disable blending
  - Blending is disabled, by default
  - Blending is not a cheap operation and should only be enabled for the geometry that needs it.
  - When you are done rendering that geometry, you should disable blending.
  - Also, try to batch triangles that use blending and render them at once, so that you can avoid turning blending on and off multiple times per frame.

Blending Equation

- SpriteBatch class – enabling/disabling blending

```
spriteBatch.Begin()
spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.Opaque)
spriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.Additive)
spriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend)
```

- BasicEffect class – enabling/disabling blending

```
// enabling
GraphicsDevice.BlendState = BlendState.AlphaBlend;
GraphicsDevice.DepthStencilState = DepthStencilState.None;
// draw semi-transparent object
basicEffect.Alpha = 0.5f;
drawTransparentObject();
// disabling
GraphicsDevice.BlendState = BlendState.Opaque;
GraphicsDevice.DepthStencilState = DepthStencilState.Default;
```

Blend Factors

- Source and destination blend factors are specified

http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphics.blend(v=xnagamestudio.40)

Blend.Zero	(0, 0, 0, 0)
Blend.One	(1, 1, 1, 1)
Blend.SourceColor	(r_s, g_s, b_s, a_s)
Blend.InverseSourceColor	(1-r_s, 1-g_s, 1-b_s, 10a_s)
Blend.SourceAlpha	(a_s, a_s, a_s, a_s) srcFactor의 default
Blend.InverseSourceAlpha	(1-a_s, 1-a_s, 1-a_s, 1-a_s) dstFactor의 default
Blend.DestinationAlpha	(a_d, a_d, a_d, a_d)
Blend.InverseDestinationAlpha	(1-a_d, 1-a_d, 1-a_d, 1-a_d)
Blend.DestinationColor	(r_d, g_d, b_d, a_d)
Blend.InverseDestinationColor	(1-r_d, 1-g_d, 1-b_d, 1-a_d)
Blend.SourceAlphaSaturation	(f, f, f, 1), f = min(a_s, 1-a_d)

Blend Factors

```
// set the source & destination blend factors directly in an effect file
pass P0
{
    vertexShader = compile vs_2_0 VS();
    pixelShader = compile vs_2_0 PS();

    AlphaBlendEnable = true;
    SrcBlend = One;
    DestBlend = One;
    BlendOp = RevSubtract;
};
```

Blend Factors

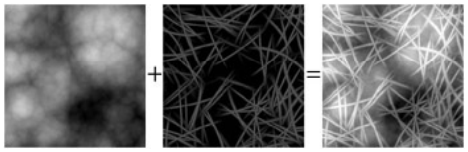
```
// alpha blending
pass P0
{
    vertexShader = compile vs_2_0 VS();
    pixelShader = compile vs_2_0 PS();

    // Use these states to blend RGB components
    AlphaBlendEnable = true;
    SrcBlend = One;
    DestBlend = One;
    BlendOp = RevSubtract;

    // Use these states to blend the alpha components
    SeparateAlphaBlendEnable = true;
    SrcBlendAlpha = SrcAlpha;
    DestBlendAlpha = InvSrcAlpha;
    BlendOpAlpha = Add;
};
```

Blend Factors Example 1

- No blending
  - Want to keep the original destination pixel exactly as it is and not overwrite or blend it with the source pixel currently being rasterized
  - Set the source pixel blend factor to ZERO and the destination pixel blend factor to ONE.
  - *OutputPixel* = *SourcePixel* ⊗ *SourceBlendFactor* + *DestPixel* ⊗ *DestBlendFactor*
  - *OutputPixel* = *SourcePixel* ⊗ (0, 0, 0, 0) + *DestPixel* ⊗ (1, 1, 1, 1)
  - *OutputPixel* = (0, 0, 0, 0) + *DestPixel*
  - *OutputPixel* = *DestPixel*

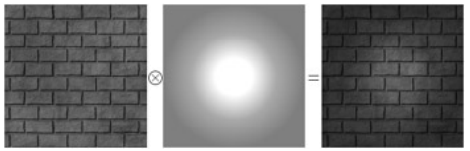
Blend Factors Example 2

- Additive blending
  - Want to directly add them together to form a new image
  - Set the source pixel blend factor to ONE and the destination pixel blend factor to ONE.
  - *OutputPixel* = *SourcePixel* ⊗ *SourceBlendFactor* + *DestPixel* ⊗ *DestBlendFactor*
  - *OutputPixel* = *SourcePixel* ⊗ (1, 1, 1, 1) + *DestPixel* ⊗ (1, 1, 1, 1)
  - *OutputPixel* = *SourcePixel* + *DestPixel*



Blend Factors Example 3

- Multiply blending
  - Want to multiply a source pixel with its corresponding destination pixel
  - Set the source pixel blend factor to ZERO and the destination pixel blend factor to SRCCOLOR.
  - *OutputPixel* = *SourcePixel* ⊗ *SourceBlendFactor* + *DestPixel* ⊗ *DestBlendFactor*
  - *OutputPixel* = *SourcePixel* ⊗ (0, 0, 0, 0) + *DestPixel* ⊗ *SourcePixel*
  - *OutputPixel* = *DestPixel* ⊗ *SourcePixel*



Blend Factors Example 4

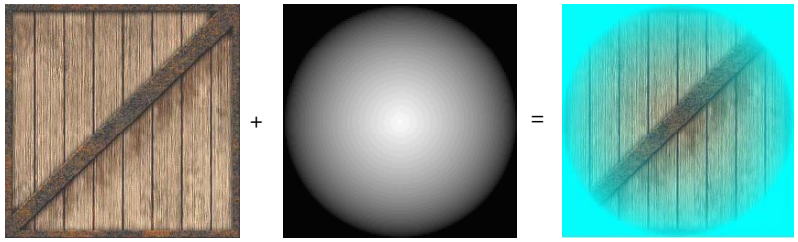
- Alpha blending
  - Want to blend the source and destination pixels based on the transparency percent of the source pixel
  - set the source pixel blend factor to SRCALPHA and the destination pixel blend factor to INVSRCALPHA.
  - *OutputPixel* = *SourcePixel* ⊗ *SourceBlendFactor* + *DestPixel* ⊗ *DestBlendFactor*
  - *OutputPixel* = *SourcePixel* ⊗ (s_a, s_a, s_a, s_a) + *DestPixel* ⊗ (1 - s_a, 1 - s_a, 1 - s_a, 1 - s_a)
  - *OutputPixel* = s_a · *SourcePixel* + (1 - s_a) · *DestPixel*

Transparency

- Transparency
  - To add alpha information to a texture, we create a fourth channel called the alpha channel (transparency).
  - Alpha channel [0, 255]:
    opacity: alpha 0 - 0% transparent
    alpha 128 - 50% half transparent
    alpha 255 - 100% opaque
  - If we want to use the texture alpha to set transparency of object, set Alpha Blending.

Alpha Channel

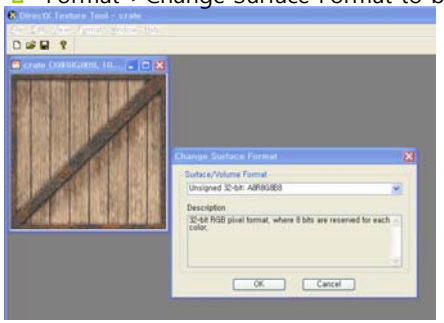
- Alpha channel
  - Alpha value: (1) calculate it during shading process, (2) obtain it from the texture alpha channel.
  - Alpha channel (RGBA): By using a texture we can control the transparency of an object at the pixel level, and we can have very complicated patterns of alpha values.



RGB channel　　Alpha channel　　Result

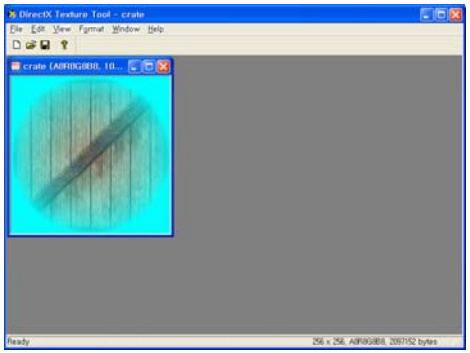
Creating Alpha Channel Using DX Tex Tool

- Create a DDS image file (containing alpha channel)
  - Run DirectX9 SDK Texture Tool
  - Run Program-> Microsoft DirectX 9.0 SDK Update (Feb 2010)-> DirectX Utilities-> DirectX Texture Tool
  - Add the alpha channel
    - File->Open <crate.jpg> (originally, 24-bit RGB)
    - Format->Change Surface Format to be 32-bit A8R8G8B8로 변경



Creating Alpha Channel Using DX Tex Tool

- Create a DDS image file (containing alpha channel)
  - Add the alpha channel in the image file
    - Prepare for 8 bit grey-scale image (e.g., alphachannel.bmp)
    - Load alphachannel.bmp (8-bit grey-scale) by File->Open Onto Alpha Channel Of This Texture
    - Save 'createwalpha.dds' by File->Save As



Render Semi-Transparency

- Render a semi-transparent object

Render Semi-Transparency

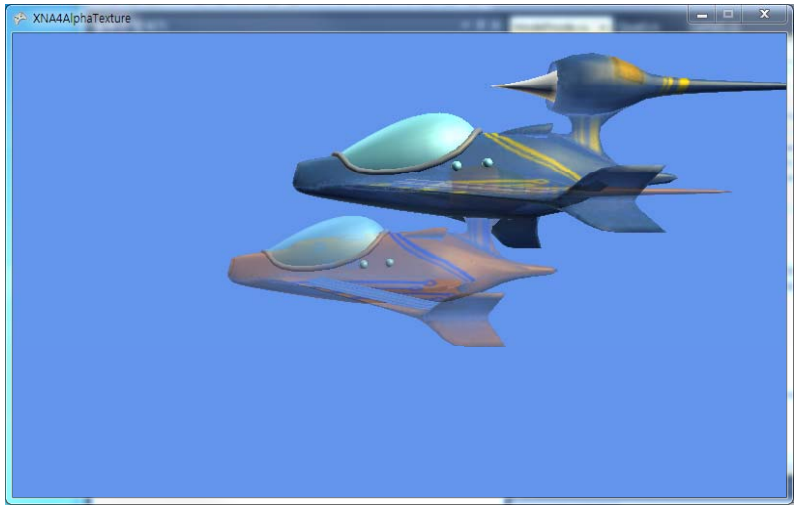
```
mTeapotMtrl.ambient = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);
mTeapotMtrl.diffuse = D3DXCOLOR(1.0f, 1.0f, 1.0f, 0.5f);
mTeapotMtrl.spec = D3DXCOLOR(0.8f, 0.8f, 0.8f, 1.0f);
mTeapotMtrl.specPower = 16.0f;

// In vertex shader we just pass along the alpha component to
// the pixel shader.
outVS.diffuse.a = gDiffuseMtrl.a;

// In pixel shader we just set the pixel's alpha value to the
// interpolated color's alpha component.
float4 DirLightTexPS(float4 c : COLOR0, float4 spec : COLOR1, float2 tex0 :
    TEXCOORD0) : COLOR
{
    float3 texColor = tex2D(TexS, tex0).rgb;
    float3 diffuse = c.rgb * texColor;
    return float4(diffuse + spec.rgb, c.a);
}
```

Render Semi-Transparency

```
protected override void Draw(GameTime gameTime)
{
... // 중간생략
    // Enable alpha blending.
    GraphicsDevice.BlendState = BlendState.AlphaBlend;
    GraphicsDevice.DepthStencilState = DepthStencilState.None;

    // [...] Set effect parameters
    basicEffect.Alpha = 0.5f;
    // draw cube with BasicEffect
    drawCube();

    // Disable alpha blending.
    GraphicsDevice.BlendState = BlendState.Opaque;
    GraphicsDevice.DepthStencilState = DepthStencilState.Default;
}
```

Render Semi-Transparency

- Render a semi-transparent 3D model



Render Semi-Transparency

```
private void DrawModel(Model m) {
    // save current states
    BlendState bs = GraphicsDevice.BlendState;
    DepthStencilState dss = GraphicsDevice.DepthStencilState;
    // set for transparencies
    GraphicsDevice.BlendState = BlendState.AlphaBlend;
    GraphicsDevice.DepthStencilState = DepthStencilState.Default;
    GraphicsDevice.SamplerStates[0] = SamplerState.LinearWrap;
    // draw 3D model with BasicEffect
    Matrix[] transforms = new Matrix[m.Bones.Count];
    m.CopyAbsoluteBoneTransformsTo(transforms);
    foreach (ModelMesh mesh in m.Meshes)      {
        foreach (BasicEffect effect in mesh.Effects) {
                effect.EnableDefaultLighting();
                effect.Alpha = 0.5f;
```

TeapotDemo

```
                effect.View = view;
                effect.Projection = projection;
                effect.World =
    Matrix.CreateFromYawPitchRoll(MathHelper.ToRadians(90), 45.0f, 0.0f) *
    Matrix.CreateTranslation(0, 0, zoom) * Matrix.CreateScale(0.002f, 0.002f,
    0.002f) * transforms[mesh.ParentBone.Index] *
    Matrix.CreateTranslation(modelPosition);
            }
            mesh.Draw();
        }

    // reset blending states
    GraphicsDevice.BlendState = bs;
    GraphicsDevice.DepthStencilState = dss;
}
```