# Advanced Texturing

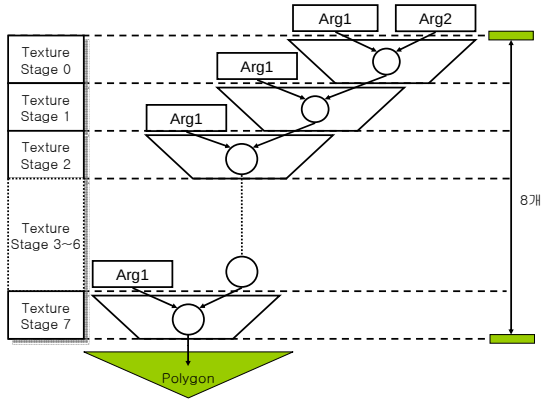
305890
Spring 2010
5/7/2010
Kyoung Shin Park

## Overview

- Multi-texturing
- Light Mapping
- Billboarding
- Texture Animation & Sprites

## Multi-Texturing

- Multi-texturing
  - Refers to the technique of combining several texture maps together to form a net texture.



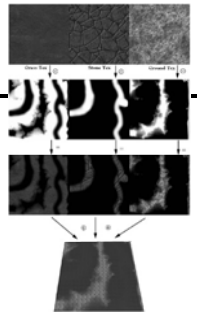

## Multi-Texturing

- Check the multi-texturing support

```
Device->GetDeviceCaps(&DeviceCaps);
if(!(DeviceCaps.MaxSimultaneousTextures > 1))
{
   MessageBox(0,"single pass multitexturing not supported!“,0,0);
   return false;
}
```

## Multi-Texturing

- Multi-texturing blending map



Let $C_{1,ij}, C_{2,ij}, C_{3,ij}$ be colors sampled from the three texture maps for the ijth pixel

Let $B_{ij} = (B_{r,ij}, B_{g,ij}, B_{b,ij})$

The final color of the ijth pixel $F_{ij} = w_1 C_{1,ij} + w_2 C_{2,ij} + w_3 C_{3,ij}$

where $w_1 = \frac{B_{r,ij}}{B_{r,ij} + B_{g,ij} + B_{b,ij}}, w_2 = \frac{B_{g,ij}}{B_{r,ij} + B_{g,ij} + B_{b,ij}}, w_3 = \frac{B_{b,ij}}{B_{r,ij} + B_{g,ij} + B_{b,ij}}$

## Multi-Texturing

- Creating and enabling the textures

```
IDirect3DTexture9*   mTex0; // Texture 0(grass)
IDirect3DTexture9*   mTex1; // Texture 1(stone)
IDirect3DTexture9*   mTex2; // Texture 2(ground)
IDirect3DTexture9*   mBlendMap; // Blend Texture(blend map)
D3DXCreateTextureFromFile(gd3dDevice, “grass0.dds”, &mTex0);
D3DXCreateTextureFromFile(gd3dDevice, “stone2.dds”, &mTex1);
D3DXCreateTextureFromFile(gd3dDevice, “ground0.dds”, &mTex2);
D3DXCreateTextureFromFile(gd3dDevice, “blendmap.jpg”,
    &mBlendMap);
mFX->SetTexture(mhTex0, mTex0);
mFX->SetTexture(mhTex1, mTex1);
mFX->SetTexture(mhTex2, mTex2);
mFX->SetTexture(mhBlendMap, mBlendMap);
```

## Multi-Texturing

- Sampler objects

```
uniform extern texture gTex0;
uniform extern texture gTex1;
uniform extern texture gTex2;
uniform extern texture gBlendMap;
Sampler Tex0S = sampler_state {
    Texture = <gTex0>;
    MinFilter = Anisotropic;
    MapFilter = LINEAR;
    MipFilter = LINEAR;
    MaxAnisotropy = 8;
    AddressU = WRAP;
    AddressV = WRAP;
};
```

## Multi-Texturing

- Vertex & Pixel Shader

```
struct OutputVS {
    float4 posH          : POSITION0;
    float4 diffuse       : COLOR0;
    float4 spec          : COLOR1;
    float2 tiledTexC     : TEXCOORD0;
    float2 nonTiledTexC  : TEXCOORD1;
};
```

## Multi-Texturing

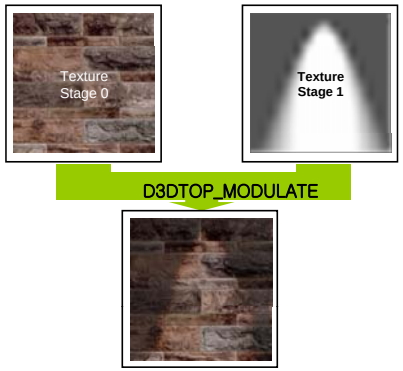
```
OutputVS TerrainMultiTexVS(float3 posL: POSITION0,
        float3 normalL: NORMAL0, float2 tex0: TEXCOORD0) {
    OutputVS outVS = (OutputVS) 0;
    // transform normal to world space
    float3 normalW = mul(float4(normalL, 0.0f), gWorldInvTrans).xyz;
    normalW = normalize(normalW);
    float3 posW = mul(float4(posL, 1.0f), gWorld).xyz;
    // Lighting Calculations Omitted...
    outVS.diffuse.rgb = ambient + diffuse;
    outVS.diffuse.a = gDiffuseMtrl.a;
    outVS.spec = float4(spec, 0.0f);
    outVS.posH = mul(float4(posL, 1.0f), gWVP);
    // Pass on texture coordinates to be interpolated in rasterization.
    outVS.tiledTexC = tex0 * 16.0f; // Scale tex-coord to
    outVS.nonTiledTexC = tex0; // tile 16 times.
    return outVS;
};
```

## Multi-Texturing

```
float4 TerrainMultiTexPS(float4 diffuse: COLOR0, float4 spec: COLOR1,
      float2 tiledTexC: TEXCOORD0, float2 nonTiledTexC: TEXCOORD1) : COLOR {
    float3 c0 = tex2D(Tex0S, tiledTexC).rgb;
    float3 c1 = tex2D(Tex1S, tiledTexC).rgb;
    float3 c2 = tex2D(Tex2S, tiledTexC).rgb;
    // blend map is not tiled.
    float3 B = tex2D(BlendMapS, nonTiledTexC).rgb;
    // find inverse of all the blend weights to scale the total color to range [0, 1].
    float totalInverse = 1.0f / (B.r + B.g + B.b);
    // scale colors by each layer by its corresponding weight stored in blend map.
    c0 *= B.r * totalInverse;
    c1 *= B.g * totalInverse;
    c2 *= B.b * totalInverse;
    // sum the colors and modulate with the lighting color
    float3 final = (c0 + c1 + c2) * diffuse.rgb;
    return float4(final + spec, diffuse.a);
};
```

## Light Mapping

- Light mapping (Single pass multi-texturing)
  - Multi-texture refers to the act of mixing two or more texture maps to create a new texture map.
  - A light map is multiplied with a texture map to produce shadows




## Light Mapping

- Direct3D supports the blending of up to 8 textures on a primitive in a single pass through the use of texture stages

```
void Render()
{
   // 스테이지0 설정 – 색상연산과 알파연산은 값을 변경없이 그대로 사용
   Device->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_SELECTARG1);
   Device->SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);
   Device->SetTextureStageState(0, D3DTSS_ALPHAOP, D3DTOP_SELECTARG1);
   Device->SetTextureStageState(0, D3DTSS_ALPHAARG1, D3DTA_TEXTURE);
   Device->SetTexture(0, g_pTex0); // wall texture

   // 스테이지1 설정 – 색상연산은 현재 텍스쳐색상과 이전 스테이지 결과(즉, 벽)를
   // MODULATE으로 곱하고, 알파연산은 사용안함
   Device->SetTextureStageState(1, D3DTSS_COLOROP, D3DTOP_MODULATE);
   Device->SetTextureStageState(1, D3DTSS_COLORARG1, D3DTA_TEXTURE);
   Device->SetTextureStageState(1, D3DTSS_COLORARG2, D3DTA_CURRENT);
   Device->SetTextureStageState(1, D3DTSS_ALPHAOP, D3DTOP_DISABLE);
   Device->SetTexture(1, g_pTex1); // light map texture

  // 중략..
}
```

## Multi Pass Multi-Texturing

- Multi-pass multi-texturing, can be used for the device that doesn't support single pass multi-texturing
  - Single Pass Multi-Texturing: draw scene (or primitives) using multiple textures
  - Multi Pass Multi-Texturing: draw scene (or primitives) multiple times
  - The disadvantage of multi-pass multi-texturing is that each primitives must be rendered as many times as you want textures on it.
  - Besides, there aren't as many operations possible as with single pass multi-texturing.

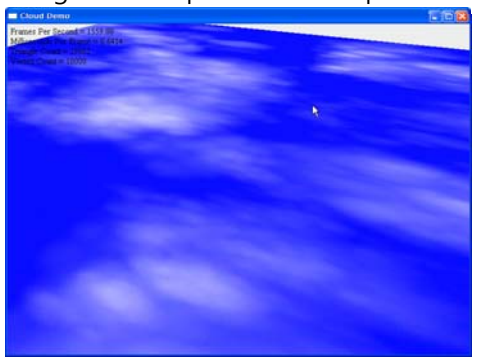
## Multi Pass Multi-Texturing

- Draw primitives multiple times with blending modulation

```
void Render()
{
  //enable blending operations
  Device->SetRenderState(D3DRS_ALPHABLENDENABLE,true);
  //standard blend factors
  Device->SetRenderState(D3DRS_SRCBLEND,D3DBLEND_ONE);
  Device->SetRenderState(D3DRS_DESTBLEND,D3DBLEND_ZERO);
  //set base texture
  Device->SetTexture(0, g_pTex0);
  Device->SetStreamSource(0,pQuadVB,0,sizeof(D3DVERTEX));
  Device->DrawPrimitive(D3DPT_TRIANGLESTRIP,0,2);
  // blending factors
  Device->SetRenderState(D3DRS_SRCBLEND,D3DBLEND_ZERO);
  Device->SetRenderState(D3DRS_DESTBLEND,D3DBLEND_SRCCOLOR);
  // set lightmap texture
  Device->SetTexture(0, g_pTex1);
  Device->SetStreamSource(0,pQuadVB,0,sizeof(D3DVERTEX));
  Device->DrawPrimitive(D3DPT_TRIANGLESTRIP,0,2);
}
```

## Texture Animation

- CloudDemo
  - Shows texture animation by scrolling 2 cloud textures at different velocities over a grid and add them together along with a blue color
  - Simply pass the texture coordinates onto the rasterization stage for interpolation and input into the pixel shader.



## Texture Animation

```
float4 CloudsPS(float4 c: COLOR0,
        float2 tex0: TEXCOORD0, float2 tex1: TEXCOORD1) : COLOR
{
   float3 c0 = tex2D(CloudS0, tex0).rgb;
   float3 c1 = tex2D(CloudS1, tex1).rgb;
   float3 blue = float3(0.0f, 0.0f, 1.0f);
   return float4(c0 + c1 + blue, 1.0f);
}
```

## Texture Animation

```
void CloudDemo::updateScene(float dt) {
    // Other update code removed...
    // Update texture coordinate offsets.
    mTexOffset0 += D3DXVECTOR2(0.11f, 0.05f) * dt;
    mTexOffset1 += D3DXVECTOR2(0.25f, 0.1f) * dt;
    // Textures repeat every 1.0, so reset back down to 0
    if(mTexOffset0.x >= 1.0f || mTexOffset0.x <= -1.0f)
        mTexOffset0.x = 0.0f;
    if(mTexOffset1.x >= 1.0f || mTexOffset1.x <= -1.0f)
        mTexOffset1.x = 0.0f;
    if(mTexOffset0.y >= 1.0f || mTexOffset0.y <= -1.0f)
        mTexOffset0.y = 0.0f;
    if(mTexOffset1.y >= 1.0f || mTexOffset1.y <= -1.0f)
        mTexOffset1.y = 0.0f;
}
```

## Sprite-Based Graphics

- Sprites
  - In computer graphics or games, a sprite is a 2D image or animation that is integrated into a larger scene.
  - In general, 2D game figures are all referred to as sprites.
  - Originally invented as a method of quickly compositing several images together in 2D video games.
  - Used in main character, enemies, any living things, projectiles (rockets, bullets, arrows, rocks, etc), vehicles, etc
- Advantages
  - No graphics hardware required
  - High quality, also on low resolution
  - Relatively easy to use and control
- Problems
  - Animation speed difficult to control
  - Lost of memory required
  - Fixed viewpoint

## Texture Animation

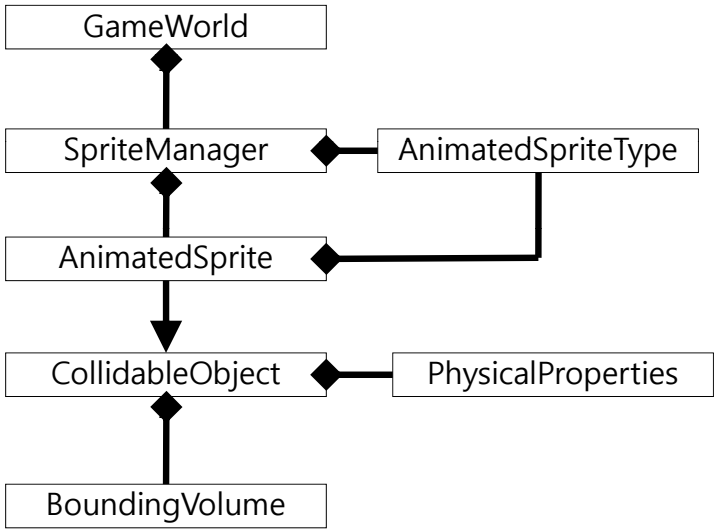
```
uniform extern float4x4 gWVP;
uniform extern texture gCloudTex0;
uniform extern texture gCloudTex1;
uniform extern float2  gTexOffset0;
uniform extern float2  gTexOffset1;

sampler CloudS0 = sampler_state {
    Texture = <gCloudTex0>;
    MinFilter = Anisotropic;
    MagFilter = LINEAR;
    MipFilter = LINEAR;
    MaxAnisotropy = 8;
    AddressU  = WRAP;
    AddressV  = WRAP;
};
```

## Texture Animation

```
struct OutputVS {
    float4 posH : POSITION0;
    float2 tex0 : TEXCOORD0;
    float2 tex1 : TEXCOORD1;
};
OutputVS CloudsVS(float3 posL : POSITION0, float2 tex0: TEXCOORD0)
{
    OutputVS outVS = (OutputVS)0;
    // Transform to homogeneous clip space.
    outVS.posH = mul(float4(posL, 1.0f), gWVP);
    // Pass on texture coordinates to be interpolated in rasterization.
    outVS.tex0 = tex0 + gTexOffset0;
    outVS.tex1 = tex0 + gTexOffset1;
    // Done--return the output.
    return outVS;
}
```

## Sprite-Based Game




## Sprite-Based Game

```
class SpriteManager
{
private:
    vector<AnimatedSpriteType*> *spriteTypes;
    vector<AnimatedSprite*> *sprites;
    AnimatedSprite *player;
```

## Sprite-Based Game

```
class AnimatedSprite : public CollidableObject
{
private:
    AnimatedSpriteType *spriteType;
    int alpha;
    int currentState;
    int currentFrame;
    int frameIndex;
    int animationCounter;
```

## Sprite-Based Game

```
class AnimatedSpriteType
{
private:
    int spriteTypeID;
    vector<vector<int>*> *animationSequences;
    vector<string*> *animationSequencesNames;
    int animationSpeed;
    vector<int> *textureIDs;
    int textureHeight, textureWidth;
```

## Sprite-Based Game

```
class CollidableObject
{
protected:
    bool currentlyCollidable;
    BoundingVolume *bv;
    PhysicalProperties *pp;
```

## Sprite-Based Game

```
class PhysicalProperties
{
protected:
    float buoyancy;
    float mass;
    bool  collidable;
    float coefficientOfRestitution;
    float x, y, z;
    float velocityX, velocityY, velocityZ;
    float accelerationX, accelerationY, accelerationZ;
```