Advanced Texturing

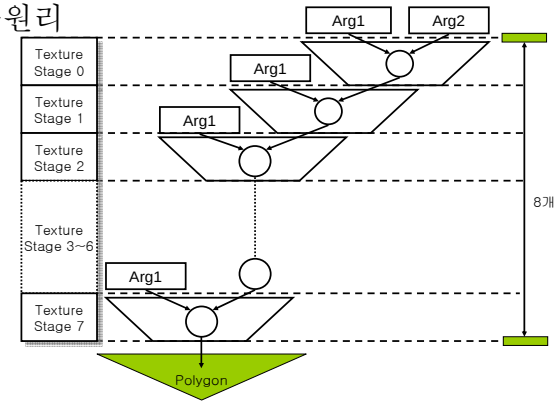
305890
2009년 봄학기
4/15/2009
박경신

Overview

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

Multi-Texturing

- 멀티 텍스처(multi texture)란 말 그대로 다중 텍스처 (여러 장의 텍스처) 즉, 하나의 폴리곤이 그려질 때 여러 장의 텍스처가 중첩되어 출력되는 것을 말한다.
- 작동원리



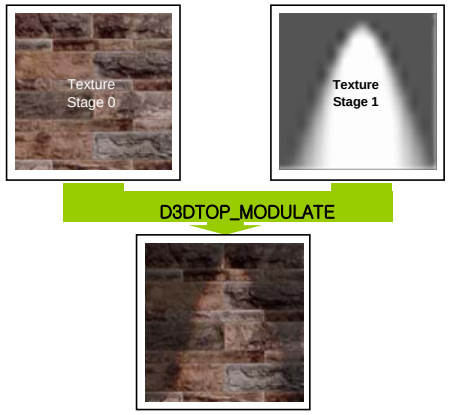
Multi-Texturing

- 멀티 텍스처링(Multitexturing) 지원 여부 확인

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

Light Mapping

- 라이트 매핑 (Light mapping)
  - 실제로 광원을 생성해서 벽면에 비추는 것이 아니라, 광원이 있는 곳에는 광원이 빛을 비추는 모양의 광원 텍스처를 벽면 텍스처와 합성해 주는 Single Pass Multitexturing 기법



Light Mapping

- 2개의 텍스처변수 선언

```
LPDIRECT3DTEXTURE9    g_pTex0    = NULL; // Texture 0(벽면)
LPDIRECT3DTEXTURE9    g_pTex1    = NULL; // Texture 1(라이트맵)
```

- Vertex 선언

```
struct Vertex {
    float _x, _y, _z;          // position
    float _nx, _ny, _nz;       // normal
    float _u1, _v1;            // first texture coordinates
    float _u2, _v2;            // second texture coordinates
// 중략
};
const DWORD Vertex::FVF = D3DFVF_XYZ | D3DFVF_NORMAL |
    D3DFVF_TEX2; // multitexture
```

Light Mapping

- 텍스처 읽어들이기

```
HRESULT InitTexture()
{
 // 벽면
 if(FAILED(D3DXCreateTextureFromFile(Device, "env.bmp", &g_pTex0)))
      return E_FAIL;

// 라이트맵
 if(FAILED(D3DXCreateTextureFromFile(Device, "lite.bmp", &g_pTex1)))
      return E_FAIL;

   return S_OK;
}
```

Light Mapping

- Render() 함수로 텍스처 합성 처리

```
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);

  // 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);

 // 중략..
}
```

Multi Pass Multi-Texturing

- 다중 패스 다중 텍스쳐링이란 다중 텍스쳐링을 지원하지 않는 하드웨어에서 다중 텍스쳐링을 흉내내는 기법
  - Single Pass Multi-Texturing은 하나의 렌더링 패스 안에서 텍스쳐를 여러 개 입히는 것
  - Multi Pass Multi-Texturing은 장면이나 다각형 자체를 여러 번 렌더링하는 것
  - 때문에, 다중 패스 다중 텍스쳐링은 여러 번 다각형을 그려야 하는 단점이 있다.
  - 또한, 싱글 패스 다중 텍스쳐링만큼 다양한 블렌딩 기능이 지원되지 않는다.

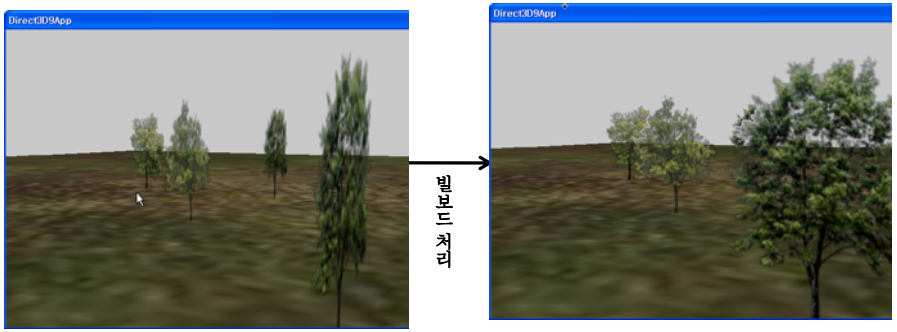
Multi Pass Multi-Texturing

- Render() 함수로 텍스처 합성 처리

```
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);
}
```

Billboarding

- 빌보드는 카메라가 어느 방향에서 바라보아도 항상 카메라의 정면을 향하고 있다.
  - 만약 우리가 사각형 메시를 빌보드가 아닌 단순 메시로 그리면 왼쪽 그림과 같이 출력될 것이다.
  - 오른쪽은 빌보드 처리 후 화면이다.



Billboarding

- 빌보드 원리
  - 빌보드의 역할은 메시의 정면이 항상 카메라를 향하도록 하는 것이다. 그 얘기는 항상 카메라를 향하도록 하는 행렬을 만들면 된다. 카메라 변환 행렬의 성분 중에서 회전행렬부분만 역변환 하면 빌보드 행렬을 구할 수 있다.
  - 회전 행렬의 정의는 다음과 같으므로, 카메라의 회전행렬 중에서 Y축 회전행렬 값을 뽑는다.

X축 회전행렬	Y축 회전행렬	Z축 회전행렬
$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

  - 이렇게 뽑아낸 Y축 회전행렬의 역행렬을 구한다. 이 역행렬을 출력하려는 빌보드 메시의 변환행렬로 사용한다.

Billboarding

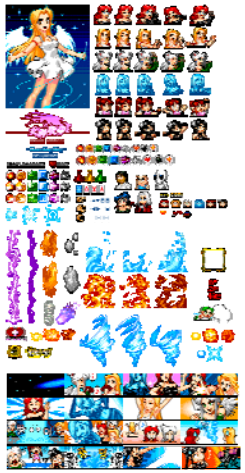
```
void DrawBillboard()
{
    // 알파채널을 사용해서 투명텍스처 효과를 낸다
    Device->SetRenderState( D3DRS_ALPHABLENDENABLE,   TRUE );
    Device->SetRenderState( D3DRS_SRCBLEND,  D3DBLEND_SRCALPHA );
    Device->SetRenderState( D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA );
    Device->SetRenderState( D3DRS_ALPHATESTENABLE, TRUE );
    Device->SetRenderState( D3DRS_ALPHAREF, 0x08 );
    Device->SetRenderState( D3DRS_ALPHAFUNC, D3DCMP_GREATEREQUAL );

    // 빌보드 행렬을 구한다
    D3DXMATRIX matView, matBillboard;
    D3DXMatrixIdentity( &matBillboard );
    Device->GetTransform( D3DTS_VIEW, &matView );
    D3DXMatrixInverse( &matBillboard, NULL, &matView ); // billboarding matrix
```

Billboarding

```
// 0번에 빌보드 텍스처를 올리고 좌표를 바꿔가며 찍는다
    for ( int z = 0 ; z <= 20 ; z += 5 ) {
      for( int x = 0 ; x <= 20 ; x += 5 ) {
          // set the tree position matrix on top of Billboard matrix
          matBillboard._41 = (float) (x - 20 + locations[z * 5 + x]);
          matBillboard._42 = 0;
          matBillboard._43 = (float) (z - 20);

          Device->SetTexture( 0, TreeTex[(x+z)%3] );
          Device->SetTransform( D3DTS_WORLD, &matBillboard );
          Device->DrawPrimitiveUP( D3DPT_TRIANGLESTRIP, 2, vtx, sizeof(Vertex)
          }
    }

    Device->SetRenderState( D3DRS_ALPHATESTENABLE, FALSE );
    Device->SetRenderState( D3DRS_ALPHABLENDENABLE, FALSE );
}
```

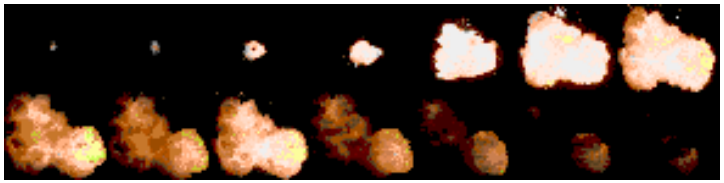
Sprites

- 화면 위에서 유령(sprite)처럼 둥둥 떠다니는 것
  - 투명색
    - 찍히지 않는 색
    - 그림과 구별이 잘 되어야 함
    - 이미지에 사용되지 않아야 함
  - 잔상이 남지 않는 그림
  - 흔히들 일반적으로 게임에 쓰이는 모든 그림을 스프라이트 (sprite) 라고 부르는 경향이 있음
- 스프라이트 용도
  - Main character, enemies, any living things, projectiles (rockets, bullets, arrows, rocks, etc), vehicles, etc



Animating Sprites

- Animating Sprites는 각각 다른 스프라이트 이미지를 화면 위에 보여주어 움직임을 표현하는 것
- Animation must be:
  - Tied to timer
  - Tied to movement (for main character)



Animating Sprites

```
LPD3DXSPRITE       gSprite = 0;    // the D3DX sprite interfaces
IDirect3DTexture9*  mainTex  = 0;
IDirect3DTexture9*  spriteTex  = 0;

typedef struct _SpriteInfo {
    RECT srcRect;
    D3DXVECTOR3 pos;   // position
    int numFrames;           // number of animation frames
    int currFrame;           // current frame
    float lastUpdate;
} SpriteInfo;

SpriteInfo sprites[5];
const int SPRITE_NUM_FRAMES_X = 3;     // num of anim frames in X
const int SPRITE_NUM_FRAMES_Y = 3;     // num of anim frame in Y
const int SPRITE_FRAME_WIDTH = 96;     // frame's width
const int SPRITE_FRAME_HEIGHT = 96;    // frame's height
const int SPRITE_FRAME_NUM = 8;        // num of anim frames
const int SPRITE_MAX_NUM = 5;          // num of animated sprites
const int SPRITE_INTERVAL = 500;       // update time interval
```

Animating Sprites

```
bool Setup() {
     // create sprite
    HRESULT hr = D3DXCreateSprite( Device, &gSprite );
     // init sprites
    for (int i = 0; i < SPRITE_MAX_NUM; i++) {
            sprites[i].srcRect.top = 0;
            sprites[i].srcRect.left = 0;
            sprites[i].srcRect.bottom = sprites[i].srcRect.top + SPRITE_FRAME_HEIGHT;
            sprites[i].srcRect.right = sprites[i].srcRect.left + SPRITE_FRAME_WIDTH;
            sprites[i].pos = D3DXVECTOR3((float) (rand() % 500), (float) (rand() % 400),0.0);
            sprites[i].currFrame = 0;                          // 현재 보여지는 frame index
            sprites[i].numFrames = SPRITE_FRAME_NUM;    // frame 개수
            sprites[i].lastUpdate = (float)timeGetTime();
    }
    // load textures
    D3DXCreateTextureFromFile( Device, "background.dds", &mainTex );
    D3DXCreateTextureFromFileEx(Device, "goblinetiles.png", D3DX_DEFAULT_NONPOW2,
                                        D3DX_DEFAULT_NONPOW2, 1, 0,
                                        D3DFMT_UNKNOWN, D3DPOOL_DEFAULT,
                                        D3DX_DEFAULT, D3DX_DEFAULT, 0xFFFFFFFF,
                                        NULL, NULL, &spriteTex);
… }
```

Animating Sprites

```
bool Display() {

    if (Device) {

        // updates
        float currTime  = (float)timeGetTime();
        for (int i = 0; i < SPRITE_MAX_NUM; i++) {
            if (currTime - sprites[i].lastUpdate > SPRITE_INTERVAL) {
            // increment the sprite animation frame
            // if you have reached the last frame, reset to the first frame
                    sprites[i].currFrame++;
                    if (sprites[i].currFrame > sprites[i].numFrames - 1)
                        sprites[i].currFrame = 0;
            // update the last updated time
            sprites[i].lastUpdate = currTime;
            }
        }

…// 중간생략
```

Animating Sprites

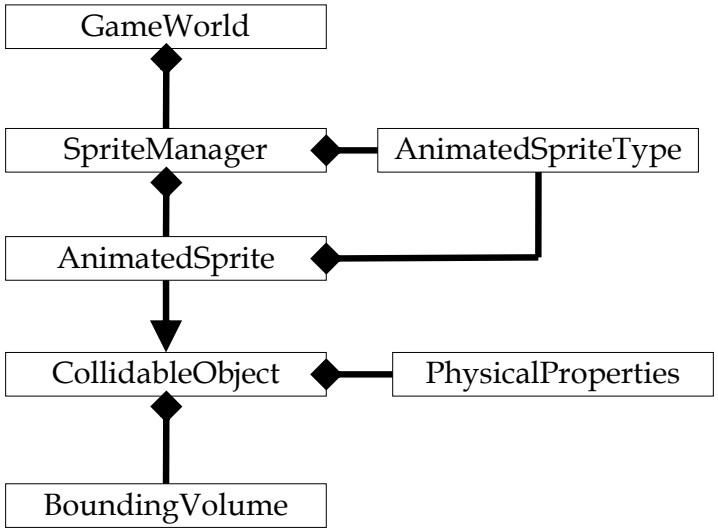
```
    // draw sprite with alpha blending
    gSprite->Begin(D3DXSPRITE_ALPHABLEND);

… // 중간생략

    // draw animated sprites
    for (int i = 0; i < SPRITE_MAX_NUM; i++) {
        unsigned currFrameX = sprites[i].currFrame % SPRITE_NUM_FRAMES_X;
        unsigned currFrameY = sprites[i].currFrame / SPRITE_NUM_FRAMES_Y;
        RECT destRect;          // Create a temporary destination RECT
        destRect.left = currFrameX * SPRITE_FRAME_WIDTH;
        destRect.right = destRect.left + SPRITE_FRAME_WIDTH;
        destRect.top = currFrameY * SPRITE_FRAME_HEIGHT;
        destRect.bottom = destRect.top + SPRITE_FRAME_HEIGHT;
        // This is the other method of using the sprite interface.
        gSprite->Draw(spriteTex, &destRect, NULL, &sprites[i].pos, 0xFFFFFFFF);
    }
    // finished drawing.
    gSprite->End();
….
}
```

Animating Sprites

Animating Sprites

- 애니메이션 스프라이트를 화면에 보여주려면 무슨 데이터가 필요한가?
  - 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

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;
```