Flexible Camera

305890
2009년 봄학기
5/20/2009
박경신

Building a Flexible Camera Class

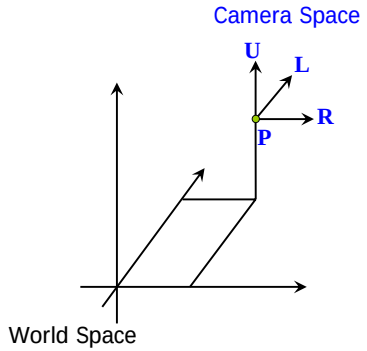
- Camera Design
- Implementation Details
- Camera 예제

Camera Design

- 구현동기
  - 고정된 카메라 위치 설정을 위해서 D3DXMatrixLookAtLH( ) 함수 사용
  - 장점: 고정된 위치에 카메라를 놓고 목표 지점을 겨냥
  - 단점: 사용자 입력에 반응하여 카메라를 이동
- 목표
  - 비행 시뮬레이션이나 1인칭 시점의 게임(e.g. First-Person Shooting game)에 적합한 유연한 Camera 클래스 구현

Camera Design

- 4개의 카메라 벡터
  - world coordinate system에서의 카메라의 위치와 방향을 정의하기 위해 사용
  - 우향 (right), 상향 (up), 전방 (look), 위치 벡터 (position vector)
    - P = Position vector
    - R = Right vector
    - U = Up vector
    - L = Look vector
  - (**R, U, L**)
    - 방위 벡터 (orientation vectors)
    - 정직교 (orthonormal)
    - 직교 행렬 (orthogonal matrix)
    - 역행렬 == 전치행렬



Camera Design

- 구현하고자 하는 카메라 동작 – 6 가지의 자유도 (6DOF)
  - Pitch - 우행 벡터를 기준으로 회전
  - Yaw - 상향 벡터를 기준으로 회전
  - Roll - 전방 벡터를 기준으로 회전
  - Strafe - 우향 벡터 방향으로 이동 (옆걸음질)
  - Fly - 상향 벡터 방향으로 이동 (날기)
  - Walk - 전방 벡터 방향으로 이동 (전,후진)
- 구현하고자 하는 카메라 타입
  - AIRCRAFT – 6가지 자유도를 허용
  - LANDOBJECT – 특정 축으로의 이동을 제한
    - 1인칭 슈팅게임 등과 같이 주인공이 하늘을 날 수 없도록 제한함

Camera Design

```
#include <d3dx9.h>
class Camera {
public:
    enum CameraType { LANDOBJECT, AIRCRAFT };
    Camera();
    Camera(CameraType cameraType);
    ~Camera();
    void strafe(float units); // left/right
    void fly(float units);    // up/down
    void walk(float units);   // forward/backward
    void pitch(float angle); // rotate on right vector
    void yaw(float angle);   // rotate on up vector
    void roll(float angle);  // rotate on look vector
    void getViewMatrix(D3DXMATRIX* V);
    void setCameraType(CameraType cameraType);
    void getPosition(D3DXVECTOR3* pos);
    void setPosition(D3DXVECTOR3* pos);
    void getRight(D3DXVECTOR3* right);
    void getUp(D3DXVECTOR3* up);
    void getLook(D3DXVECTOR3* look);
private:
    CameraType  _cameraType;
    D3DXVECTOR3 _right;
    D3DXVECTOR3 _up;
    D3DXVECTOR3 _look;
    D3DXVECTOR3 _pos;
};
```

Camera Design

```
#include "camera.h"

Camera::Camera() {
    _cameraType = AIRCRAFT;
    _pos = D3DXVECTOR3(0.0f, 0.0f, 0.0f);
    _right = D3DXVECTOR3(1.0f, 0.0f, 0.0f);
    _up = D3DXVECTOR3(0.0f, 1.0f, 0.0f);
    _look = D3DXVECTOR3(0.0f, 0.0f, 1.0f);
}

Camera::Camera(CameraType cameraType) {
    _cameraType = cameraType;
    _pos = D3DXVECTOR3(0.0f, 0.0f, 0.0f);
    _right = D3DXVECTOR3(1.0f, 0.0f, 0.0f);
    _up = D3DXVECTOR3(0.0f, 1.0f, 0.0f);
    _look = D3DXVECTOR3(0.0f, 0.0f, 1.0f);
}

Camera::~Camera() {}

void Camera::getPosition(D3DXVECTOR3* pos) {*pos = _pos; }
void Camera::setPosition(D3DXVECTOR3* pos) { _pos = *pos; }
void Camera::getRight(D3DXVECTOR3* right) {*right = _right; }
void Camera::getUp(D3DXVECTOR3* up) {*up = _up; }
void Camera::getLook(D3DXVECTOR3* look) {*look = _look; }
void Camera::setCameraType(CameraType cameraType) {_cameraType = cameraType; }
```

View Matrix

- 주어진 카메라 vector로부터 View Matrix (뷰 행렬)를 변환을 위하여 World space의 점 q를 View space의 점 q′으로 변환하는 행렬 V를 구해야 한다.
  - $q' = qV$
- World space에서 View space로의 변환 (viewing transformation)
  - 카메라 위치가 World Space의 원점에 오도록 이동시킴 (카메라와 모든 물체)
  - Look vector가 +z축과 일치되도록 회전시킴 (카메라와 모든 물체)
- 4개의 카메라 벡터
  - $p = (p_x, p_y, p_z)$
  - $r = (r_x, r_y, r_z)$
  - $u = (u_x, u_y, u_z)$
  - $l = (l_x, l_y, l_z)$

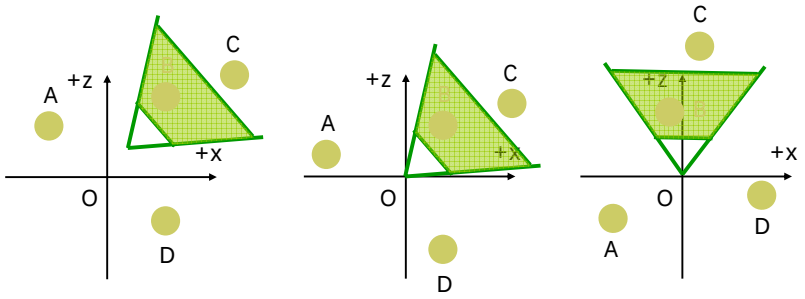
View Matrix

- View space transformation – 변환 행렬 **V**

p = (p_x, p_y, p_z) : position vector
r = (r_x, r_y, r_z) : right vector
u = (u_x, u_y, u_z) : up vector
l = (l_x, l_y, l_z) : look vector

⇒

pV = *(0, 0, 0)* : ***origin***
rV = *(1, 0, 0)* : ***x-axis***
uV = *(0, 1, 0)* : ***y-axis***
lV = *(0, 0, 1)* : ***z-axis***



View Matrix Transformation: Translation

- **p** 를 원점으로 이동

$$\mathbf{x} - \mathbf{p} = 0 \qquad (x \quad y \quad z) - (p_x \quad p_y \quad p_z) = (0 \quad 0 \quad 0)$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -p_x & -p_y & -p_z & 1 \end{bmatrix}$$

$$\mathbf{xT} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -p_x & -p_y & -p_z & 1 \end{bmatrix} = \begin{bmatrix} x - p_x & y - p_y & z - p_z & 1 \end{bmatrix}$$

View Matrix Transformation: Rotation

- 3개 카메라 벡터 (**r, u, l**)를 월드의 축에 일치 하도록 회전

$$\mathbf{rR} = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{uR} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{lR} = \begin{bmatrix} l_x & l_y & l_z \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

View Matrix Transformation: Rotation

- 3개 카메라 벡터 (**r, u, l**)를 월드의 축에 일치 하도록 회전

$$\mathbf{rR} = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{uR} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{lR} = \begin{bmatrix} l_x & l_y & l_z \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

⇒

$$\underset{\mathbf{M}}{\begin{bmatrix} r_x & r_y & r_z \\ u_x & u_y & u_z \\ l_x & l_y & l_z \end{bmatrix}} \underset{\mathbf{R}}{\begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}} = \underset{\mathbf{I}}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}$$

$$\mathbf{MR} = \mathbf{I} \Rightarrow \mathbf{R} = \mathbf{M}^{-1}$$

$$\mathbf{R} = \mathbf{M}^{-1} = \mathbf{M}^{T} = \begin{bmatrix} r_x & u_x & l_x \\ r_y & u_y & l_y \\ r_z & u_z & l_z \end{bmatrix}$$

View Matrix Transformation: V

- 뷰 변환 행렬 **V**:

$$\mathbf{V} = \mathbf{TR} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -p_x & -p_y & -p_z & 1 \end{bmatrix} \begin{bmatrix} r_x & u_x & l_x & 0 \\ r_y & u_y & l_y & 0 \\ r_z & u_z & l_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

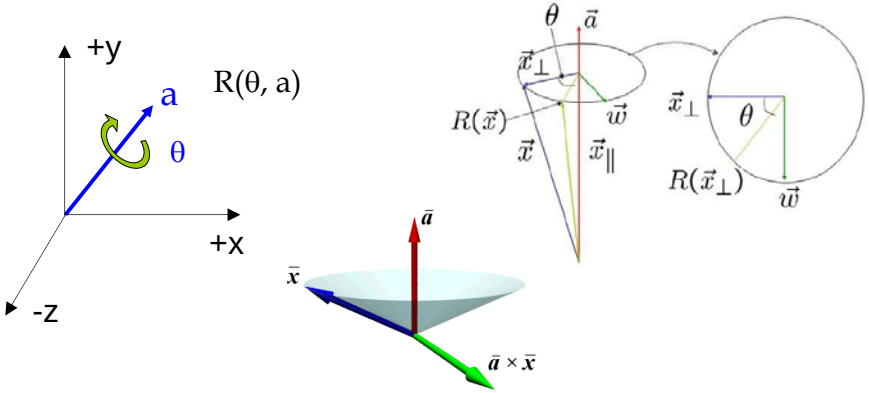
$$= \begin{bmatrix} r_x & u_x & l_x & 0 \\ r_y & u_y & l_y & 0 \\ r_z & u_z & l_z & 0 \\ -\mathrm{p}\cdot\mathrm{r} & -\mathrm{p}\cdot\mathrm{u} & -\mathrm{p}\cdot\mathrm{l} & 1 \end{bmatrix}$$

Camera :: getViewMatrix ( )

```
void Camera::getViewMatrix(D3DXMATRIX* V)
{
    // Keep camera's axes orthogonal to each other
    D3DXVec3Normalize(&_look, &_look);

    D3DXVec3Cross(&_up, &_look, &_right);
    D3DXVec3Normalize(&_up, &_up);

    D3DXVec3Cross(&_right, &_up, &_look);
    D3DXVec3Normalize(&_right, &_right);

    // Build the view matrix:
    float pr = -D3DXVec3Dot(&_right, &_pos);
    float pu = -D3DXVec3Dot(&_up, &_pos);
    float pl = -D3DXVec3Dot(&_look, &_pos);

    (*V)(0, 0) = _right.x; (*V)(0, 1) = _up.x;  (*V)(0, 2) = _look.x;  (*V)(0, 3) = 0.0f;
    (*V)(1, 0) = _right.y; (*V)(1, 1) = _up.y; (*V)(1, 2) = _look.y;  (*V)(1, 3) = 0.0f;
    (*V)(2, 0) = _right.z; (*V)(2, 1) = _up.z; (*V)(2, 2) = _look.z;  (*V)(2, 3) = 0.0f;
    (*V)(3, 0) = pr;          (*V)(3, 1) = pu;      (*V)(3, 2) = pl;          (*V)(3, 3) = 1.0f;
}
```

Rotation Axis/Angle

- 임의의 회전축 (axis)에 대한 하나의 회전각도 (angle) 4개의 숫자로 표현한다. 임의의 회전축을 나타내는 단위벡터 a (x, y, z)와 단위 벡터 주위로 회전각도를 나타내는 θ (0~360)값으로 구성된다.



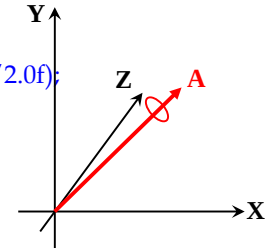
Yaw, Pitch, Roll

- 임의의 축을 기준으로 회전하는 변환행렬

```
D3DXMATRIX &D3DXMatrixRotationAxis (
        D3DXMATRIX *pOut,     // output rotation matrix
        CONST D3DXVECTOR *pV, // arbitrary axis
        FLOAT Angle);   // angle of rotation (in radians)
```

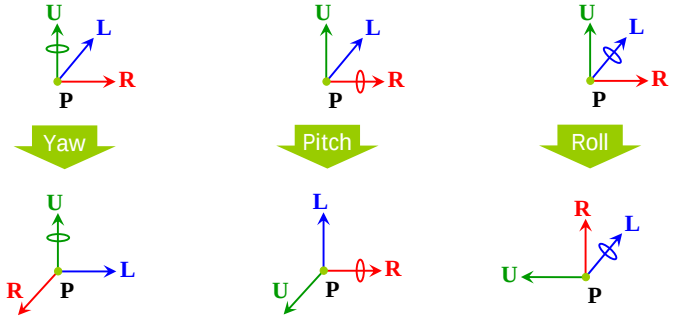
  - 예제: vector (0.707, 0.707, 0)로 정의된 축을 기준으로 pi/2만큼 회전

```
D3DXMATRIX R;
D3DXVECTOR3 axis(0.707f, 0.707f, 0.0f);
D3DXMatrixRotationAxis(&R, &axis, D3DX_PI/2.0f);
```



Yaw, Pitch, Roll

- Yaw/pitch/roll – up/right/look vector를 기준으로 회전
- LANDOBJECT에 대해서는 물체가 기울어진 상태에서 yaw/roll은 부자연스러움
  - Yaw는 up vector가 아닌 y-축을 기준으로 회전
  - Roll은 이용할 수 없도록 함



Camera :: pitch ( ) and yaw ( )

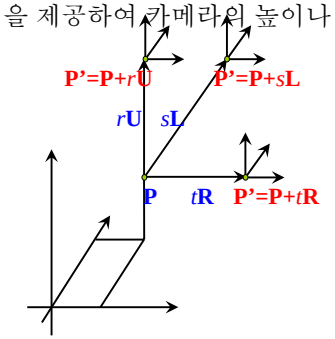
```
void Camera::pitch(float angle)
{
    D3DXMATRIX T;
    D3DXMatrixRotationAxis(&T, &_right, angle);

    // rotate _up and _look around _right vector
    D3DXVec3TransformCoord(&_up,&_up, &T);
    D3DXVec3TransformCoord(&_look,&_look, &T);
}

void Camera::yaw(float angle)
{
    D3DXMATRIX T;

    // rotate around world y (0, 1, 0) always for land object
    if( _cameraType == LANDOBJECT )
        D3DXMatrixRotationY(&T, angle);

    // rotate around own up vector for aircraft
    if( _cameraType == AIRCRAFT )
        D3DXMatrixRotationAxis(&T, &_up, angle);

    // rotate _right and _look around _up or y-axis
    D3DXVec3TransformCoord(&_right,&_right, &T);
    D3DXVec3TransformCoord(&_look,&_look, &T);
}
```

Camera :: roll ( )

```
void Camera::roll(float angle)
{
    // only roll for aircraft type
    if( _cameraType == AIRCRAFT )
    {
        D3DXMATRIX T;
        D3DXMatrixRotationAxis(&T, &_look,   angle);

        // rotate _up and _right around _look vector
        D3DXVec3TransformCoord(&_right,&_right, &T);
        D3DXVec3TransformCoord(&_up,&_up, &T);
    }
}
```

Walk, Strafe, Fly

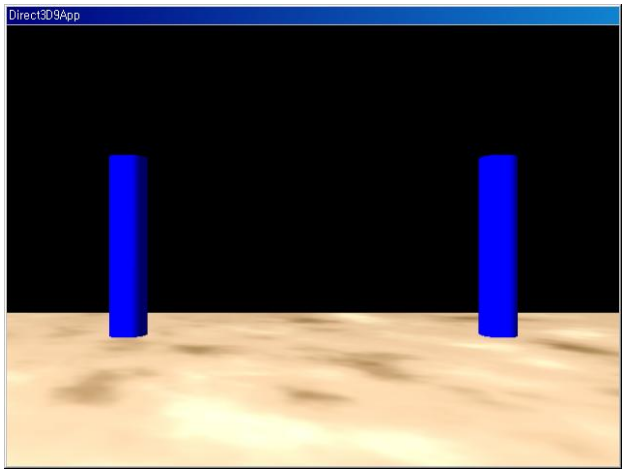
- Strafe/Fly/Walk – up/right/look vector를 기준으로 이동
  - 이동하고자 하는 크기/방향의 벡터를 더하면 됨
- LANDOBJECT:
  - xz평면으로 움직임을 제한하여야 함. 위를 보는 상황에서 전진하거나 기울이진 상황에서 옆걸음을 해도 공중에 뜨지 않도록 함.
  - 계단/언덕을 오르는 방법으로 고도가 바뀔 수 있으므로 Camera::setPosition 방법을 제공하여 카메라의 높이나 위치 지정이 가능하도록 함



Camera :: walk, strafe, fly ( )

```
void Camera::walk(float units)
{
    // move only on xz plane for land object
    if( _cameraType == LANDOBJECT )
        _pos += D3DXVECTOR3(_look.x, 0.0f, _look.z) * units;
    if( _cameraType == AIRCRAFT )
        _pos += _look * units;
}

void Camera::strafe(float units)
{
    // move only on xz plane for land object
    if( _cameraType == LANDOBJECT )
        _pos += D3DXVECTOR3(_right.x, 0.0f, _right.z) * units;
    if( _cameraType == AIRCRAFT )
        _pos += _right * units;
}

void Camera::fly(float units)
{
    // move only on y-axis for land object
    if( _cameraType == LANDOBJECT )
        _pos.y += units;
    if( _cameraType == AIRCRAFT )
        _pos += _up * units;
}
```

Example: Camera



Control Keys

- W / S – 전진 / 후진 (Walk forward/backward)
- A / D – 왼쪽 / 오른쪽 옆걸음질 (Strafe left/right)
- R / F – 위 / 아래 날기 (Fly up / down)
- Left / Right arrow keys – Yaw
- Up / Down arrow keys – Pitch
- N / M – Roll

Example: DrawBasicScene

- d3dUtility.h

```
bool DrawBasicScene(
        IDirect3DDevice9* device, // pass in 0 for cleanup
        float scale); // uniform scale

struct Vertex {
    Vertex() {}
    Vertex(float x, float y, float z, float nx, float ny, float nz, float u, float v)
    {
        _x = x; _y = y; _z = z;
        _nx = nx; _ny = ny; _nz = nz;
        _u = u; _v = v;
    }
    float _x, _y, _z;
    float _nx, _ny, _nz;
    float _u, _v;
    static const DWORD FVF;
}
```

Example: DrawBasicScene

```
bool d3d::DrawBasicScene(IDirect3DDevice9* device, float scale) {
    static IDirect3DVertexBuffer* floor = 0;
    static IDirect3DTexture9* tex = 0;
    static ID3DXMesh* pillar = 0;
    HRESULT hr = 0;
    if (device == 0) {
        if (floor && tex && pillar) {
            d3d::Release<IDirect3DVertexBuffer9*>(floor);
            d3d::Release<IDirect3DTexture9*>(tex);
            d3d::Release<ID3DXMesh*>(pillar);
        } else if (!floor && !tex && !pillar) {
            device->CreateVertexBuffer(6 * sizeof(d3d::Vertex), 0,
                        d3d::Vertex::FVF, D3DPOOL_MANAGED, &floor, 0);
            Vertex* v = 0;
            floor->Lock(0, 0, (void**) &v, 0);
            v[0] = Vertex(-20.0f, -2.5f, -20.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f);
            v[1] = Vertex(-20.0f, -2.5f,  20.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f);
            v[2] = Vertex( 20.0f, -2.5f,  20.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f);
             ….
            v[5] = Vertex(-20.0f, -2.5f, -20.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f);
            floor->Unlock();
```

Example: DrawBasicScene

```
            D3DXCreateCylinder(device, 0.5f, 0.5f, 5.0f, 20, 20, &pillar, 0)
            D3DXCreateTextureFromFile(device, "desert.bmp", &tex)
        } else {
            device->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
            device->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
            device->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_POINT);
            D3DXVECTOR3 dir(0.707f, -0.707f, 0.707f);
            D3DXCOLOR col(1.0f, 1.0f, 1.0f, 1.0f);
            D3DLIGHT9 light = d3d::InitDirectionalLight(&dir, &col);
            device->SetLight(0, &light);
            device->LightEnable(0, true);
            device->SetRenderState(D3DRS_NORMALIZENORMALS, true);
            device->SetRenderState(D3DRS_SPECULARENABLE, true);
            // render
            D3DXMATRIX T, R, P, S;
            D3DXMatrixScaling(&S, scale, scale, scale);
            D3DXMatrixRotationX(&R, -D3DX_PI *0.5f);
```

Example: DrawBasicScene

```
            // draw floor
            D3DXMatrixIdentity(&T);
            T = T * S;
            device->SetTransform(D3DTS_WORLD, &T);
            device->SetMaterial(&d3d::WHITE_MTRL);
            device->SetTexture(0, tex);
            device->SetStreamSorce(0, floor, 0, sizeof(Vertex));
            device->SetFVF(Vertex::FVF);
            device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);
            // draw pillar
            device->SetMaterial(&d3d::BLUE_MTRL);
            device->SetTexture(0, 0);
            for (int i = 0; I < 5; i++) {
                D3DXMatrixTranslation(&T, -5.0f, 0.0f, -15.0f + (i * 7.5f));
                T = T * S;
                device->SetTransform(D3DTR_WORLD, &P);
                pillar->DrawSubset(0);
                D3DXMatrixTranslation(&T, 5.0f, 0.0f, -15.0f + (i * 7.5f));
                P = R * T * S;
                device->SetTransform(D3DTR_WORLD, &P);
                pillar->DrawSubset(0);
            }
    }   return true; }
```

Example: Camera

```
#include "d3dHelper.h"
#include "d3dUtility.h"
#include "camera.h"
IDirect3DDevice9* Device = 0;
const int Width = 640;
const int Height = 480;

Camera TheCamera(Camera::LANDOBJECT);

bool Setup() {
    d3d::DrawBasicScene(Device, 0.0f);
    // projection matrix
    D3DXMATRIX proj;
    D3DXMatrixPerspectiveFovLH(&proj, D3DX_PI * 0.25f, (float) Width/ (float)
                                        Height, 1.0f, 1000.0f);
    Device->SetTransform(D3DTS_PROJECTION, &proj);
    return true;
}
void Cleanup() {
    // pass 0 for the first parameter to clean up
    d3d::DrawBasicScene(0, 0.0f);
}
```

Example: Camera

```
bool Display(float timeDelta) {
    if (Device) {
        if (::GetAsyncKeyState('W') & 0x8000f) TheCamera.walk( 4.0f*timeDelta);
        if (::GetAsyncKeyState('S') & 0x8000f) TheCamera.walk( -4.0f*timeDelta);
        if (::GetAsyncKeyState('A') & 0x8000f) TheCamera.strafe( -4.0f*timeDelta);
        if (::GetAsyncKeyState('D') & 0x8000f) TheCamera.strafe( 4.0f*timeDelta);
        if (::GetAsyncKeyState('R') & 0x8000f) TheCamera.fly( 4.0f*timeDelta);
        if (::GetAsyncKeyState('F') & 0x8000f) TheCamera.fly( -4.0f*timeDelta);
        if (::GetAsyncKeyState(VK_UP) & 0x8000f)
                                    TheCamera.pitch( 1.0f*timeDelta);
        if (::GetAsyncKeyState(VK_DOWN) & 0x8000f)
                                    TheCamera.pitch( -1.0f*timeDelta);
        if (::GetAsyncKeyState(VK_LEFT) & 0x8000f)
                                    TheCamera.yaw( -1.0f*timeDelta);
        if (::GetAsyncKeyState(VK_RIGHT) & 0x8000f)
                                    TheCamera.yaw( 1.0f*timeDelta);
        if (::GetAsyncKeyState('N') & 0x8000f)
                                    TheCamera.roll( 1.0f*timeDelta);
        if (::GetAsyncKeyState('M') & 0x8000f)
                                    TheCamera.roll( -1.0f*timeDelta);
```

Example: Camera

```
        // update the view matrix representing the camera
        D3DXMATRIX V;
        TheCamera.getViewMatrix(&V);
        Device->SetTransform(D3DTS_VIEW, &V);
        // render
        Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0x00000000, 1.0f, 0);
        Device->BeginScene();
        d3d::DrawBasicScene(Device, 1.0f);
        Device->EndScene();
        Device->Present(0, 0, 0, 0);
        }
        return true;
}
```