Cameras

Le but de ce TD est de mettre en place différents modes de caméra utilisés en jeu vidéo. Il est basé sur le projet "Camera", réalisé par Frank Luna, qui est une application DirectX qui permet de bouger la caméra en fonction des actions de l'utilisateur.
La classe Camera définie permet de faire bouger la caméra en mode AIRCRAFT ou en mode LANDOBJECT.

Instructions :

1) Implémentez une classe CameraManager qui permette de changer le mode de caméra en cours de jeu (sans interpolation pour le moment).
   - M pour changer de mode

2) Implémentez une classe CameraDebug qui permette de déplacer la caméra en mode debug.
   - Up/Down: Pitch up/down
   - Right/Left: Yaw Right/Left
   - Z/S: Move Front/Back
   - Q/D: Strafe Right/Left

3) Implémentez une classe CameraFirstPerson qui permette de déplacer la caméra en first person.
   - Up/Down: Pitch up/down
   - Right/Left: Yaw Right/Left
   - Z/S: Move Front/Back on horizontal plane
   - Q/D: Strafe Right/Left

4) Faites en sorte que le cube puisse être controlé par le clavier et qu'une camera le suive(classe CameraThirdPerson).
   - Up/Down: Pitch up/down
   - Right/Left: Yaw Right/Left
   - Z/S: Move Front/Back

5) Ajoutez une interpolation entre chaque changement de mode en utilisant les quaternions.

Documentation :

IDirect3DDevice9::SetTransform Method

Sets a single device transformation-related state.

Syntax :

```
HRESULT SetTransform(
  [in]  D3DTRANSFORMSTATETYPE State,
  [in]  const D3DMATRIX *pMatrix
);
```

Parameters :

State [in]

D3DTRANSFORMSTATETYPE

Device-state variable that is being modified. This parameter can be any member of the **D3DTRANSFORMSTATETYPE** enumerated type, or the D3DTS_WORLDMATRIX macro.

pMatrix [in]

D3DMATRIX

Pointer to a D3DMATRIX structure that modifies the current transformation.

Return Value :

HRESULT

If the method succeeds, the return value is D3D_OK. D3DERR_INVALIDCALL is returned if one of the arguments is invalid.

D3DXQUATERNION Structure

Describes a quaternion.

Syntax :

```
typedef struct D3DXQUATERNION {
  FLOAT x;
  FLOAT y;
  FLOAT z;
  FLOAT w;
} D3DXQUATERNION, *LPD3DXQUATERNION;
```

Members :

x

FLOAT

The x-component.

y

FLOAT

The y-component.

z

FLOAT

The z-component.

w

FLOAT

The w-component.

D3DXMATRIX Structure

A 4x4 matrix that contains methods and operator overloads.

Syntax :

```
typedef struct D3DXMATRIX {
  FLOAT _ij;
} D3DXMATRIX, *LPD3DXMATRIX;
```

Members :

_ij

FLOAT

The (i, j) component of the matrix, where i is the row number and j is the column number. For example, _34 means the same as [a34], the component in the third row and fourth column.

Remarks :

C programmers cannot use the D3DXMATRIX structure, they must use the D3DMATRIX structure. C++ programmers can take advantage of overloaded constructors and assignment, unary, and binary (including equality) operators.

In D3DX, the _34 element of a projection matrix cannot be a negative number. If your application needs to use a negative value in this location, it should scale the entire projection matrix by -1 instead.

D3DXMatrixRotationQuaternion Function

Builds a rotation matrix from a quaternion.

Syntax :

```
D3DXMATRIX * D3DXMatrixRotationQuaternion(
  __inout  D3DXMATRIX *pOut,
```

```
  __in     const D3DXQUATERNION *pQ
);
```

Parameters :

pOut [in, out]
D3DXMATRIX

Pointer to the **D3DXMATRIX** structure that is the result of the operation.

pQ [in]
D3DXQUATERNION

Pointer to the source **D3DXQUATERNION** structure.

Return Value :

D3DXMATRIX

Pointer to a **D3DXMATRIX** structure built from the source quaternion.

Remarks :

The return value for this function is the same value returned in the *pOut* parameter. In this way, the **D3DXMatrixRotationQuaternion** function can be used as a parameter for another function.

For information about how to calculate quaternion values from a direction vector (x, y, z) and an angle of rotation, see **D3DXQUATERNION**.

D3DXQuaternionSlerp Function

Interpolates between two quaternions, using spherical linear interpolation.

Syntax :

```
 D3DXQUATERNION * D3DXQuaternionSlerp(
  __inout  D3DXQUATERNION *pOut,
  __in     const D3DXQUATERNION *pQ1,
  __in     const D3DXQUATERNION *pQ2,
  __in     FLOAT t
);
```

Parameters :

pOut [in, out]
D3DXQUATERNION

Pointer to the **D3DXQUATERNION** structure that is the result of the operation.

pQ1 [in]
D3DXQUATERNION

Pointer to a source **D3DXQUATERNION** structure.

pQ2 [in]
D3DXQUATERNION

Pointer to a source **D3DXQUATERNION** structure.

t [in]
FLOAT

Parameter that indicates how far to interpolate between the quaternions.

Return Value :

D3DXQUATERNION

Pointer to a **D3DXQUATERNION** structure that is the result of the interpolation.

Source : http://msdn.microsoft.com/en-us/library