

# Introdução ao Direct3D 9.0

Organizado por  
*Manuel Menezes de Oliveira Neto*

Copyright © Manuel M. Oliveira

## Estrutura do Curso

- Introdução à Programação para Windows
- Exemplo de Programa Simples usando Direct3D
- Definindo e Renderizando Vértices
- Utilizando Matrizes
- Definindo e Utilizando Fontes de Luz
- Renderizando Superfícies Semi-Transparentes
- Usando Texturas

Copyright © Manuel M. Oliveira

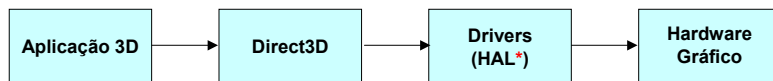
## Estrutura do Curso (Cont.)

- Exibindo Texto na Janela
- Interação via Mouse
- Usando Arquivos .X (XFiles)

Copyright © Manuel M. Oliveira

## Direct3D – Visão Geral

- **API gráfica** que permite a geração de imagens a partir de descrições de cenas 3D usando hardware gráfico
- Provê uma **camada intermediária** entre as aplicações (e.g., jogos) e o hardware gráfico



- Mantém um **conjunto de estados** que definem como a cena será renderizada
- O uso de Direct3D, requer uma aplicação Windows (Win32) com uma janela para renderizar as cenas

\* Hardware Abstraction Layer

Copyright © Manuel M. Oliveira

## Direct3D – Referências

- DirectX 9.0c SDK (Summer 2004)
  - Vários programas exemplos e tutoriais sobre uso de Direct3D
- Tutorial sobre Direct3D na web
  - Explica os exemplos que vêm com os tutoriais DirectX 9.0 SDK
  - [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9\\_c/directx/graphics/TutorialsAndSamples/Tutorials/Direct3DTutorials.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/TutorialsAndSamples/Tutorials/Direct3DTutorials.asp)
- Livro
  - **Introduction to 3D Game Programming with DirectX 9.0**, de Frank D. Luna, Wordware Publishing Inc., 2003.

Copyright © Manuel M. Oliveira

## Programação para Windows Visão Geral

- Aplicações Windows se baseiam no modelo de *programação orientada a eventos*
- Quando um evento ocorre, o Windows (SO) envia uma *mensagem* para a aplicação à qual o evento se destina
- O Windows também coloca a mensagem na *fila de mensagens* da aplicação
- A aplicação checa sua fila de mensagens constantemente (*laço de mensagens - message loop*)
- As mensagens recebidas são repassadas à rotina que trata as mensagens (*tratador de mensagens da janela*)

Copyright © Manuel M. Oliveira

## Estrutura de uma Aplicação Windows

- Includes e Variáveis Globais
  - O arquivo de cabeçalho `<windows.h>`
  - Uma variável do tipo `HWND` (handle para uma janela)
- WinMain
  - Windows equivalente da função `main()` de C/C++
  - Descreve, registra e cria a janela
  - Contém o *laço de mensagens*
- Função para Tratamento de Mensagens
  - Implementa o código faz o tratamento de mensagens

Copyright © Manuel M. Oliveira

## Estrutura de uma Aplicação Windows com Direct3D

- Includes e Variáveis Globais
  - O arquivo de cabeçalho `<d3d9.h>` substitui `<windows.h>`
- Função para inicializar a API Direct3D (e.g., `InitD3D`)
- WinMain
- Função para Tratamento de Mensagens
- Função para Renderização da Cena
- Função para Liberação de Recursos Direct3D Alocados

Copyright © Manuel M. Oliveira

## Exemplo: WinMain()

```
//-----
// Nome: WinMain()
// Desc: Ponto de Entrada da Aplicação
//-----
INT WINAPI WinMain( HINSTANCE hInst, HINSTANCE, LPSTR, INT )
{
    // Define uma classe janela e a registra junto ao Windows
    WNDCLASSEX wc = { sizeof(WNDCLASSEX), CS_CLASSDC, MsgProc, 0L, 0L, GetModuleHandle(NULL), NULL, NULL, NULL, NULL,
        "D3D Tutorial", NULL }; // define uma classe janela com as características desejadas
    RegisterClassEx( &wc ); // Registra a classe com o Windows (SO)
                                Nome da Classe
    // Cria a janela da aplicação
    HWND hWnd = CreateWindow("D3D Tutorial", "Título da Janela",
        WS_OVERLAPPEDWINDOW, 100, 100, 300, 300, GetDesktopWindow(), NULL, wc.hInstance, NULL );
                                Posição Dimensões

    // Inicializa o Direct3D
    if( SUCCEEDED( InitD3D( hWnd ) ) )
    {
        // Mostra a janela e a atualiza
        ShowWindow( hWnd, SW_SHOWDEFAULT );
        UpdateWindow( hWnd );

        // Entra no laço de mensagens
        MSG msg;
        while ( GetMessage( &msg, NULL, 0, 0 ) ) // Laço de Mensagem
        {
            TranslateMessage( &msg ); // Interpreta a mensagem do Windows
            DispatchMessage( &msg ); // Envia a mensagem recebida para MsgProc
        }
        UnregisterClass( "D3D Tutorial", wc.hInstance );
        return 0;
    }
}
```

Função para tratamento de mensagens

## Exemplo: Init3D3()

```
#include <d3d9.h>
//-----
// Variáveis Globais
//-----
LPDIRECT3D9 g_pD3D = NULL; // Objeto do tipo D3D
LPDIRECT3DDEVICE9 g_pd3dDevice = NULL; // Dispositivo de Rendering
//-----
// Nome: InitD3D()
// Desc: Inicializa o Direct3D
//-----
HRESULT InitD3D( HWND hWnd )
{
    // Cria o objeto D3D necessário para criar o dispositivo (D3DDevice).
    if( NULL == ( g_pD3D = Direct3DCreate9( D3D_SDK_VERSION ) ) )
        return E_FAIL;

    // Inicializa a estrutura usada para criar o dispositivo D3DDevice.
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory( &d3dpp, sizeof(d3dpp) );
    d3dpp.Windowed = TRUE;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;

    // Cria o dispositivo Direct3D. Usa o adaptador default e solicita o uso do hardware gráfico, mas processamento de vértices
    (T&L) por SW
    if( FAILED( g_pD3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
        D3DCREATE_SOFTWARE_VERTEXPROCESSING, &d3dpp, &g_pd3dDevice ) ) )
        return E_FAIL;
                                Ponteiro para o dispositivo gráfico associado à janela
    return S_OK;
}

Obs.: Este ponteiro será utilizado em todos os comandos da API Direct3D
```

## Exemplo: Tratamento de Mensagens

```
#!/-----  
// Nome: MsgProc()  
// Desc: Tratador de Mensagens da Janela  
//-----  
LRESULT WINAPI MsgProc( HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam )  
{  
    switch( msg )  
    {  
        case WM_DESTROY: // Mensagem gerada quando a janela é destruída  
            Cleanup(); // Função que libera os recursos alocado (definida pelo usuário)  
            PostQuitMessage( 0 ); // Mensagem para finalizar o programa  
            return 0;  
        case WM_LBUTTONDOWN: // Mensagem gerada quando o botão esquerdo do mouse é pressionado.  
            ::MessageBox(0, "Left Button", "Hello", MB_OK);  
            return 0;  
        case WM_RBUTTONDOWN: // Mensagem gerada quando o botão direito do mouse é pressionado  
            ::MessageBox(0, "Right Button", "Hello", MB_OK);  
            return 0;  
        case WM_PAINT: // Mensagem gerada quando a janela é movida ou redimensionada  
            Render(); // Função que renderiza a cena 3D (definida pelo usuário)  
            ValidateRect( hWnd, NULL );  
            return 0;  
        case WM_KEYDOWN: // Mensagem gerada quando uma tecla é pressionada  
            if ( wParam == VK_ESCAPE ) // Tecla pressionada foi o <ESC>  
                ::DestroyWindow(hWnd); // Gera mensagem WM_DESTROY para finalizar a aplicação  
            return 0;  
    }  
    return DefWindowProc( hWnd, msg, wParam, lParam ); // Função default – trata os casos não cobertos pela função MsgProc  
}
```

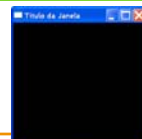
## Exemplo: Renderização da Cena

```
#!/-----  
// Nome: Render()  
// Desc: Renderiza a cena  
//-----  
VOID Render()  
{  
    if( NULL == g_pd3dDevice ) // Verifica se o dispositivo foi inicializado  
        return;  
    //  
    // Inicializa o back buffer ( neste exemplo o color buffer e o Z buffer) – Equivalente a glClear()  
    //  
    g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0,0,0) | D3DCLEAR_ZBUFFER, 1.0f, 0 );  
    //  
    // Os comandos para renderização da cena devem ficar entre os comandos BeginScene() e EndScene()  
    //  
    if( SUCCEEDED( g_pd3dDevice->BeginScene() ) ) // Equivalente a glBegin()  
    {  
        //  
        // <Comandos para renderização da cena devem ser inseridos aqui>  
        //  
        //  
        // Sinaliza o final da cena  
        //  
        g_pd3dDevice->EndScene(); // Equivalente a glEnd()  
    }  
    //  
    // Exibe o conteúdo rasterizado no back buffer  
    //  
    g_pd3dDevice->Present( NULL, NULL, NULL, NULL ); // Equivalente a glutSwapBuffers  
}
```

## Exemplo: Liberação dos Recursos Alocados

```
//-----  
// Nome: Cleanup()  
// Desc: Libera os recursos alocados  
//-----  
VOID Cleanup()  
{  
    if( g_pd3dDevice != NULL)  
        g_pd3dDevice->Release(); // libera o dispositivo  
  
    if( g_pD3D != NULL)  
        g_pD3D->Release();       // libera o objeto D3D  
}
```

## Exercícios (1)



- Crie um projeto no Visual C++ para o programa *prog\_01.cpp*
  - Prog\_01.cpp é o programa básico nos 4 últimos slides
  - No seu projeto você precisará incluir as seguintes bibliotecas:
    - d3d9.lib, d3dx9.lib
    - Arquivos .h para Direct3D estão disponíveis em C:\DXSDK\Include
    - Arquivos .lib para Direct3D estão disponíveis em C:\DXSDK\Lib
- Execute o programa Prog\_01.cpp e:
  - Clique sobre a janela com os botões esquerdo e direito do mouse
  - Pressione uma letra e estude a porção do código da função **MsgProg** que faz este tratamento (esta porção não foi incluída nos slides)
  - Pressione <Esc>
- Modifique o programa para alterar as dimensões e a cor de fundo da janela

## Definindo e Renderizando Vértices

- Primitivas Geométricas (pontos, linhas e triângulos) em Direct3D são definidas por seqüências de vértices
- Cenas 3D são criadas construindo-se modelos geométricos a partir de tais primitivas
- Para se renderizar uma cena é preciso:
  - Especificar os atributos dos vértices (e.g., coordenadas em 3D, normal, coordenadas de textura) que comporão um modelo 3D
  - Inicializar um vertex buffer a partir desta especificação
    - Um *vertex buffer* é um objeto que armazena os vértices a serem renderizados
  - Renderizar o modelo a partir do vertex buffer

Copyright © Manuel M. Oliveira

## Especificando Atributos de Vértices

- Atributos de vértices são definidos por meio de uma **estrutura** criada pelo usuário, **acompanhada de um FVF**
- FVF – Flexible Vertex Format
  - Descreve o conteúdo do vertex buffer
  - A ordem dos elementos na estrutura deve coincidir com a seqüência de “flags” no FVF

### Exemplo:

```
struct Estrutura_de_Vertice {  
    FLOAT x, y, z, rhw; // Posição do vértice já transformado, i.e., já em  
                        // coordenadas da janela em 2D - canto sup. esq. (0,0)  
    DWORD color;        // Cor do vértice.  
};
```

```
#define D3DFVF_Estrutura_de_Vertice (D3DFVF_XYZRHW | D3DFVF_DIFFUSE)
```

Copyright © Manuel M. Oliveira

## Vertex Buffer

- Um vertex buffer é criado alocando-se memória para armazenar um certo número de vértices de dado tipo

**Exemplo:** Considere a existência do vetor abaixo contendo dados de 3 vértices do tipo **Estrutura\_de\_Vertice**

```
Estrutura_de_Vertice vertices[ ] =  
{ // x,      y,      z,      rhw,   cor  
  { 150.0f, 50.0f, 0.5f, 1.0f, 0xffff0000, },  
  { 250.0f, 250.0f, 0.5f, 1.0f, 0xff00ff00, },  
  { 50.0f, 250.0f, 0.5f, 1.0f, 0xff00ffff, },  
};
```

- Neste caso, precisamos criar um vertex buffer com capacidade para armazenar 3 vértices do tipo **Estrutura\_de\_Vertice**

Copyright © Manuel M. Oliveira

## Criando um Vertex Buffer

- Criação do Vertex Buffer

```
LPDIRECT3DVERTEXBUFFER9 g_pVB = NULL; // Ponteiro para Vertex Buffer  
                                     // g_pVB decl. variável global  
...  
If ( FAILED( g_pd3dDevice->CreateVertexBuffer( 3*sizeof(Estrutura_de_Vertice),  
                                                0 /*Usage*/, D3DFVF_Estrutura_de_Vertice,  
                                                D3DPOOL_DEFAULT, &g_pVB, NULL ) ) )  
    return E_FAIL;
```

Copyright © Manuel M. Oliveira

## Inicializando um Vertex Buffer

- Bloqueia o *vertex buffer* (a partir de um offset, certo número de bytes)
- **Lock()** retorna em *pVertices* o endereço do offset especificado dentro do vertex buffer
- **memcpy** copia o conteúdo de *vertices[]* para a área apontada por *pVertices* (i.e., para o *vertex buffer*)

```
VOID* pVertices; // Apontador temporário para copiar os dados de vertices
                // do vetor vertices[ ] para o vertex buffer

if( FAILED( g_pVB->Lock( 0, sizeof(vertices), (void**)&pVertices, 0 ) ) )
    return E_FAIL;

memcpy( pVertices, vertices, sizeof(vertices) );
g_pVB->Unlock();
```

Copyright © Manuel M. Oliveira

## Renderizando o Conteúdo de um Vertex Buffer

- Define um “stream” de vértices (*SetStreamSource*)
  - Num. stream, ptr vertex buffer, offset no stream, bytes por vértice
- Especifica como os bytes no vertex buffer devem ser interpretados (*SetFVF*)
- Especifica o tipo e o número de primitivas (*DrawPrimitive*)
  - Primitiva, índice do primeiro vértice no vertex buffer, número de primitivas

```
g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0,0,0), 1.0f, 0 );
g_pd3dDevice->BeginScene();

g_pd3dDevice->SetStreamSource( 0, g_pVB, 0, sizeof(Estrutura_de_Vertice) );
g_pd3dDevice->SetFVF(D3DFVF_Estrutura_de_Vertice);
g_pd3dDevice->DrawPrimitive( D3DPT_TRIANGLELIST, 0, 1 );

g_pd3dDevice->EndScene();
g_pd3dDevice->Present( NULL, NULL, NULL, NULL );
```

Copyright © Manuel M. Oliveira

## Exercícios (2)



- Crie um projeto no Visual Studio para o programa `prog_02.cpp`
  - `Prog_02.cpp` define um vertex buffer e renderizar um triângulo utilizando os conceitos apresentados
- Modifique o programa `Prog_02.cpp` para renderizar um segundo triângulo definido pelos seguintes vértices

```
{ 50.0f, 250.0f, 0.5f, 1.0f, 0xff00ffff, }  
{ 50.0f, 50.0f, 0.5f, 1.0f, 0xff00ff00, }  
{ 150.0f, 50.0f, 0.5f, 1.0f, 0xffff0000, }
```
- Modifique o programa `Prog_02.cpp` para renderizar os vértices utilizando as seguintes primitivas ao invés de `D3DPT_TRIANGLELIST`:
  - `D3DPT_POINTLIST`
  - `D3DPT_LINELIST`
  - `D3DPT_LINESTRIP`

Copyright © Manuel M. Oliveira

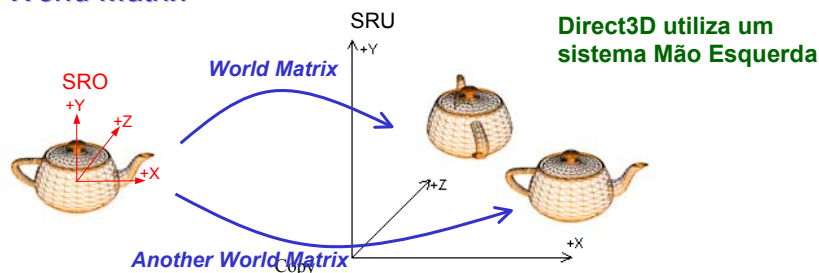
## Usando Matrizes

- Matrizes são um elemento fundamental em computação gráfica, visto que boa parte do pipeline gráfico pode ser modelado na forma de operações matriciais
- Em particular, matrizes modelam:
  - Transformação do Sistema de Referência do Objeto (SRO) para o Sistema de Referência do Universo (SRU) (*World Matrix*)
  - Transformação do Sistema de Referência do Universo para o Sistema de Referência da Câmera (SRC) (*View Matrix*)
  - Projeção (*Projection Matrix*)

Copyright © Manuel M. Oliveira

## SRO, SRU e World Matrix

- O SRO simplifica o processo de modelagem, permitindo a construção de modelos canônicos
  - Facilita o processo de instanciação dos objetos
- O mapeamento das coordenadas dos vértices do SRO para o SRU, onde as cenas são definidas, é feito pela *World Matrix*



## Instanciando uma World Matrix

```
D3DXMATRIXA16 matWorld, matTransl, matRotation; // Declara 3 matrizes 4x4
...
D3DXMatrixIdentity(&matWorld); // Inicializa a World Matrix como Identidade
...
D3DXMatrixTranslation(&matTransl, 1.0, 2.0, 3.0); // Inicializa matriz de translação
D3DXMatrixRotationY(&matRotation, fAngle); // Inicializa matriz de rotação
matWorld = matRotation * matTransl; // Compõe as operações
...
g_pd3dDevice->SetTransform( D3DTS_WORLD, &matWorld );
```

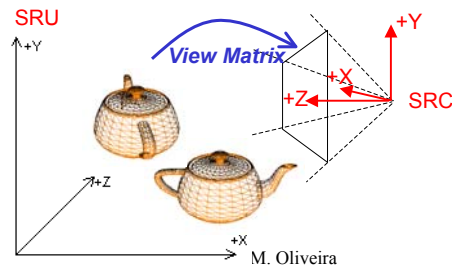
- Em Direct3D os vértices são multiplicados à esquerda

$$\begin{bmatrix} x' & y' & z' & w' \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

Copyright © Manuel M. Oliveira

## SRC e View Matrix

- O SRU permite a definição de um relacionamento espacial entre os objetos que compõem a cena
- Mas a visualização de uma cena 3D é realizada do ponto de vista da câmera (SRC)
- O mapeamento das coordenadas dos vértices do SRU para o SRC é feito pela *View Matrix*



## Instanciando a View Matrix

- O mapeamento SRU  $\rightarrow$  SRC fica determinado com:
  - posição da câmera (*vEyePt*)
  - direção de visualização (*vLookatPt* – *vEyePt*)
  - direção vertical da câmera (*UpVec*)

```
D3DXMATRIXA16 matView;           // Declara a view matrix
...
D3DXVECTOR3 vEyePt ( 0.0f, 3.0f, -5.0f );
D3DXVECTOR3 vLookatPt ( 0.0f, 0.0f, 0.0f );
D3DXVECTOR3 vUpVec ( 0.0f, 1.0f, 0.0f );

D3DXMatrixLookAtLH( &matView, &vEyePt, &vLookatPt, &vUpVec);

g_pd3dDevice->SetTransform(D3DTS_VIEW, &matView );
```

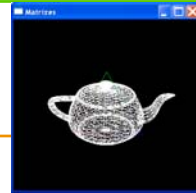
# Matriz de Projeção

- Simula uma projeção e mapeia as coordenadas de vértices do SRC para um Sistema de Referência em 2D
- A *Matriz de Projeção* fica determinada pelos:
  - Campos de visão vertical e horizontal
  - Planos de recorte próximo e distante (*near e far clipping planes*)

```
D3DXMATRIXA16 matProj;    // Declara a matriz de projeção
...
D3DXMatrixPerspectiveFovLH( &matProj, fovY_radianos, rel_aspecto, near, far );
g_pd3dDevice->SetTransform(D3DTS_PROJECTION, &matProj);
```

Copyright © Manuel M. Oliveira

## Exercícios (3)



- Abra o projeto *prog\_03*
  - Prog\_03.cpp utiliza matrizes para renderizar uma cena 3D consistindo de um triângulo em rotação em torno do eixo Y
- Modifique o programa Prog\_03.cpp para permitir que a rotação ocorra ao redor dos eixos X, Y ou Z (seleciona eixo usando tecla)
- Acrescente à cena original um teapot

```
ID3DXMesh* teapot_mesh = NULL; // Cria um ponteiro para um objeto ID3DXMesh
...
D3DXCreateTeapot(g_pd3dDevice, &teapot_mesh, 0); // cria a malha de triângulos
...
teapot_mesh->DrawSubset(0);    // Renderiza o teapot
...
if ( g_pd3dDevice != NULL )
    teapot_mesh->Release();    // Libera a malha ao final do programa
```

- Renderize os objetos usando D3DFILL\_POINT, D3DFILL\_WIREFRAME, D3DFILL\_SOLID (Utilize a tecla "W" para escolher o modo)

Copyright © Manuel M. Oliveira

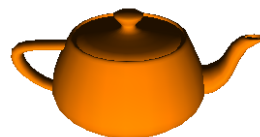
## Objetos 3D na Biblioteca D3DX

- **D3DXCreateTeapot**(g\_pd3dDevice, &malha\_teapot, 0);
- **D3DXCreateSphere**(g\_pd3dDevice, raio, subdiv1, subdiv2, &malha\_esf, &adjacente);
  - Ex.: D3DXCreateSphere(g\_pd3dDevice, 1.0f, 30, 30, &malha\_esfera, NULL);
- **D3DXCreateTorus**(g\_pd3dDevice, raio\_int, raio\_ext, subd1, subd2, &malha\_torus, &adjacente);
  - Ex.: D3DXCreateTorus(g\_pd3dDevice, 0.5f, 1.0f, 30, 30, &malha\_torus, NULL);
- **D3DXCreateCylinder**(g\_pd3dDevice, raio1, raio2, altura, subd1, subd2, &malha\_cilindro, &adjacente);
  - Ex.: D3DXCreateCylinder(g\_pd3dDevice, 0.5f, 1.0f, 2.0f, 20, 20, &malha\_cilindro, NULL);
- **D3DXCreateBox**(g\_pd3dDevice, larg, alt, prof, &malha\_paralelep, &adjacente);
  - Ex.: D3DXCreateBox(g\_pd3dDevice, 1.0f, 1.0f, 1.0f, &teapot\_mesh, NULL);



## Utilizando Fontes de Luz

- De modo a poder observar as nuances das superfícies dos objetos é preciso utilizar técnicas de sombreamento
- Entretanto, não basta apenas acrescentar uma ou mais fontes de luz. Além disso, é preciso informar:
  - as normais nos vértices dos objetos
  - as propriedades dos materiais que compõem os objetos
  - as propriedades das fontes de luz



## Informando Normais

- Para informar normais, precisamos alterar a estrutura de vértices definida anteriormente
- A cor de cada vértice é calculada automaticamente

```
struct Estrutura_de_Vertice
{
    float x, y, z;      // Coordenadas dos vértices no SRO
    float nx, ny, nz;   // Vetor normal no vértice
};
#define D3DFVF_Estrutura_de_Vertice (D3DFVF_XYZ|D3DFVF_NORMAL)
```

```
Estrutura_de_Vertice vertices[ ] =
{ // x, y, z, nx, ny, nz (no SRO)
  { -1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f },
  { 1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f },
  { 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f },
};
```

Copyright © Manuel M. Oliveira

## Propriedades dos Materiais

- Definido pelas componentes difusa e ambiental

```
D3DMATERIAL9 mtrl0;
...
ZeroMemory( &mtrl0, sizeof(D3DMATERIAL9) );
mtrl0.Diffuse.r = mtrl0.Ambient.r = 1.0f;
mtrl0.Diffuse.g = mtrl0.Ambient.g = 0.5f;
mtrl0.Diffuse.b = mtrl0.Ambient.b = 0.0f;
mtrl0.Diffuse.a = mtrl0.Ambient.a = 0.5f;      // Semi-Transparente
g_pd3dDevice->SetMaterial( &mtrl0 );
```



Copyright © Manuel M. Oliveira

## Fontes de Luz

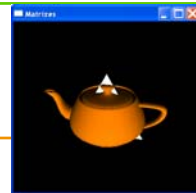
- Tipos: **Pontuais** (mais comuns), **Direcionais**, **Spotlight**
- Exemplo de Instanciação

```
D3DLIGHT9 light;
ZeroMemory( &light, sizeof(D3DLIGHT9) );           // Inicializa
light.Type    = D3DLIGHT_POINT;                     // Fonte de luz pontual
light.Position = D3DXVECTOR3(5.0f, 0.0f, -5.0f);     // Posição no SRU
light.Diffuse.r = 1.0f;                             // Luz branca
light.Diffuse.g = 1.0f;
light.Diffuse.b = 1.0f;
light.Range    = 1000.0f;                           // Alcance da fonte de luz

g_pd3dDevice->SetLight( 0, &light );                // Registra a fonte de luz com o dispositivo
g_pd3dDevice->LightEnable( 0, true );               // Liga a fonte de luz número zero
g_pd3dDevice->SetRenderState( D3DRS_LIGHTING, true ); // Habilita iluminação
g_pd3dDevice->SetRenderState( D3DRS_AMBIENT, 0x00202020 ); // Define a luz
                                                         //ambiente
```

Copyright © Manuel M. Oliveira

## Exercícios (4)



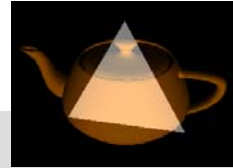
- Abra o projeto **prog\_04**
  - Prog\_04.cpp acrescenta iluminação à cena gerada no Exercício (3)
- Modifique o programa Prog\_04.cpp para ligar/desligar a iluminação ao se pressionar as teclas “L” ou “I”

```
LRESULT WINAPI MsgProc(...)
....
case WM_KEYDOWN:
...
if (wParam == 'L' || wParam == 'I') {
    Light_State = !Light_State;
    g_pd3dDevice->SetRenderState( D3DRS_LIGHTING, Light_State );
}
```

Copyright © Manuel M. Oliveira

# Superfícies Semi-Transparentes

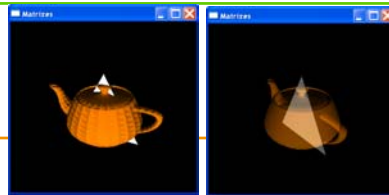
- O grau de transparência de uma superfície é definido pelo valor do canal (atributo) alfa associado às propriedades do material ou à textura aplicada a ele
- Usando Transparência



```
HRESULT InitD3D( ... )
{
    ...
    //  Habilita D3D Blending
    g_pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, true);
    //
    //  Parâmetros de Blending
    g_pd3dDevice->SetRenderState( D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
    g_pd3dDevice->SetRenderState( D3DRS_DESTBLEND, D3DBLEND_DESTALPHA);
}
```

Copyright © Manuel M. Oliveira

## Exercícios (5)



- Abra o projeto *prog\_05*
  - Prog\_05.cpp liga e desliga iluminação - Exercício (4)
- Modifique o programa Prog\_05.cpp para ligar/desligar transparência usando as teclas “B” ou “b”

```
LRESULT WINAPI MsgProc(...)
{
    ...
    case WM_KEYDOWN:
        ...
        if (wParam == 'B' || wParam == 'b') {
            Blend_State = !Blend_State;
            g_pd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, Blend_State);
        }
    }
}
```

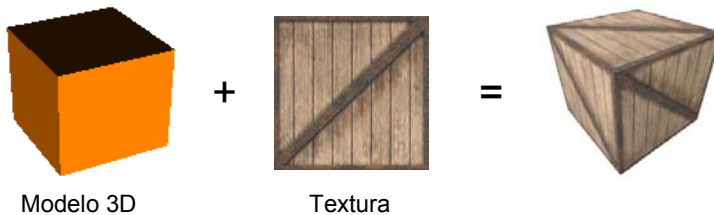
- Acrescente a possibilidade de renderizar os modelos com Shading Mode do tipo *D3DSHADE\_FLAT* ou *D3DSHADE\_GOURAUD* (use H ou h)

```
g_pd3dDevice->SetRenderState( D3DRS_SHADEMODE, Shade_mode);
```

Copyright © Manuel M. Oliveira

## Mapeamento de Texturas

- Nenhuma técnica sozinha contribui tanto para aumentar o realismo de uma cena quanto mapeamento de texturas
- Mapeamento de textura pode ser comparado a aplicação de papel parede ou de um decalque a uma superfície



Copyright © Manuel M. Oliveira

## Usando Texturas

- Para mapear texturas em polígonos, é preciso informar:
  - coordenadas de textura associadas aos vértices dos objetos
  - ler a(s) textura(s) de arquivo(s) ou gerá-las proceduralmente
  - ativar a textura antes do início da renderização das primitivas
  - desativar a textura após a renderização das primitivas
  - liberar o objeto textura ao final do programa

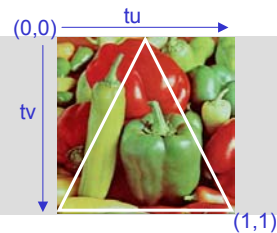
Copyright © Manuel M. Oliveira

# Especificando Coordenadas Texturas

- Requer a alteração de nossa estrutura de vértices

```
struct Estrutura_de_Vertice
{
    float x, y, z;      // Coordenadas dos vértices no SRO
    float nx, ny, nz;   // Vetor normal no vértice
    float tu, tv;       // Coordenadas de textura
};
#define D3DFVF_Estrutura_de_Vertice (D3DFVF_XYZ|D3DFVF_NORMAL|D3DFVF_TEX1)
```

```
Estrutura_de_Vertice vertices[] =
{
    // x, y, z, nx, ny, nz, tu, tv
    { -1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f },
    { 1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f },
    { 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.5f, 0.0f },
};
```



Copyright © Manuel M. Oliveira

# Carregando Textura de Arquivo

- A função `D3DXCreateTextureFromFile` permite a definição de texturas a partir de arquivos de imagens nos formatos .bmp, .dds, .jpg, .png e .tga

```
LPDIRECT3DTEXTURE9 g_pTexture = NULL; // Apontador para objeto textura
...
// Le e cria uma textura a partir de arquivo de imagem
if ( FAILED(D3DXCreateTextureFromFile(g_pd3dDevice, "peppers_256.bmp", &g_pTexture)) )
{
    MessageBox(NULL, "Não encontrou peppers_256.bmp", "Prog_06.exe", MB_OK);
    return E_FAIL;
}
```

Copyright © Manuel M. Oliveira

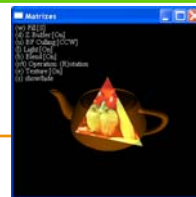
# Ativando uma Textura

- Antes de renderizar um conjunto de primitivas com uma dada textura, esta precisa ser ativada

```
g_pd3dDevice->SetTexture( 0, g_pTexture ); // Ativa a textura e a associa ao estagio 0
//
// Renderiza as primitivas no vertex buffer com a textura ativa
g_pd3dDevice->SetStreamSource( 0, g_pVB, 0, sizeof(Estrutura_de_Vertice) );
g_pd3dDevice->SetFVF( D3DFVF_Estrutura_de_Vertice );
g_pd3dDevice->DrawPrimitive( D3DPT_TRIANGLESTRIP, 0, 1 );
//
// Desativa a textura apos o rendering do conteudo do vertex buffer
g_pd3dDevice->SetTexture( 0, NULL );
...
if ( g_pTexture != NULL )
    g_pTexture->Release(); // Libera o objeto textura ao final do programa
```

Copyright © Manuel M. Oliveira

## Exercícios (6)



- Abra o projeto *prog\_06* (aplica textura no triângulo)
- Modifique o programa Prog\_06.cpp para mostrar na tela o status das opções de rendering implementadas até agora (teclas “S” ou “s”)

```
#include <d3dfont.h>
CD3DFont* Font = NULL;

// Cria um objeto do tipo font e o inicializa com as características de fonte desejada
//
Font = new CD3DFont("Times New Roman", 9, 0);
Font->InitDeviceObjects( g_pd3dDevice );
Font->RestoreDeviceObjects();

...

void show_status(void) {
//
// Mostra se o teste de Z encontra-se ativado
if (Z_Buffer_State)
    Font->DrawText(5, 15, 0xffffffff, "(d) Z Buffer:[On]");
else Font->DrawText(5, 15, 0xffffffff, "(d) Z Buffer:[Off]");
}
```

## Interação via Mouse

- Em Direct3D podemos utilizar o mouse para prover uma interface amigável para interação com a cena 3D
- Na função para tratamento de mensagens, monitoramos mensagens como as seguintes geradas pelo Windows:
  - WM\_LBUTTONDOWN / WM\_RBUTTONDOWN (Clique com botão da Esq/Dir)
  - WM\_LBUTTONUP / WM\_RBUTTONUP (Liberação do botão da Esq/Dir)
  - WM\_MOUSEMOVE (Mouse em movimento)
    - Os parâmetros desta mensagem (`wParam`) informam o estado dos botões
    - Requer testes para descobrir o estado dos botões
      - (`wParam & MK_LBUTTON`) // Testa se o botão da Esq está pressionado
      - (`wParam & MK_RBUTTON`) // Testa se o botão da Dir está pressionado

Copyright © Manuel M. Oliveira

## Interação via Mouse

- Exemplo

```
#include <Windowsx.h> // Necessária para uso das macros GET_?_LPARAM
...

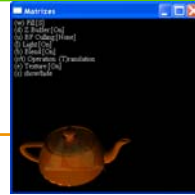
case WM_LBUTTONDOWN: // Botão da Esq foi pressionado
    PrevX = CurrentX = GET_X_LPARAM(IParam); // Coord. X do mouse na janela
    PrevY = CurrentY = GET_Y_LPARAM(IParam); // Coord. Y do mouse na janela
    return 0;
case WM_MOUSEMOVE: // Mouse em movimento
    if (wParam & MK_LBUTTON) { // Botão da Esq pressionado
        CurrentX = GET_X_LPARAM(IParam); // Coord. X atual
        CurrentY = GET_Y_LPARAM(IParam); // Coord. Y atual

        posX += 0.1f*(CurrentX-PrevX); // Atualiza coord. X de um objeto em 3D
        posY += 0.1f*(PrevY-CurrentY); // Atualiza coord. Y de um objeto em 3D

        PrevX = CurrentX; // Reseta coord. da posição anterior do mouse
        PrevY = CurrentY;
    }
    return 0;
```

Copyright © Manuel M. Oliveira

## Exercícios (7)



- Abra o projeto *prog\_07* (mostra o estado do rendering por meio de texto exibido na janela – Exercício (6))
- Modifique o programa Prog\_07.cpp para permitir a **translação** dos objetos da cena nas direções X e Y (SRU), usando **interação via mouse**
  - Utilize o código do exemplo de interação via mouse do slide anterior
  - No Prog\_07.cpp, as teclas “T” e “t” habilitam a operação de translação
  - Atualize a World Matrix utilizando o seguinte fragmento de código

```
VOID SetupMatrices()  
{  
    ...  
    D3DXMatrixTranslation(&matWorld, posX, posY, 0.0f);  
    g_pd3dDevice->SetTransform( D3DTS_WORLD, &matWorld );  
    ...  
}
```

Copyright © Manuel M. Oliveira

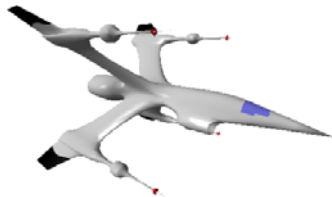
## Arquivos .X (XFiles)

- Objetos 3D são, frequentemente, criados utilizando-se modeladores 3D como 3DS Max, LightWave e Maya
- Estes programas exportam os modelos gerados (geometria, materiais, etc.) para diversos formatos
- XFile é um formato de arquivo definido especificamente para Direct3D
- A biblioteca D3DX fornece suporte a arquivos .X
- Plugins exportadores para .X estão disponíveis no DirectX9 SDK Extra – Direct3D Tools (<http://www.msdn.microsoft.com>)

Copyright © Manuel M. Oliveira

## Exemplos

- O SDK do DirectX disponibiliza vários modelos em formato .X (C:\DXSDK\Samples\Media)



bigship1.x



airplane 2.x



tiger.x

Copyright © Manuel M. Oliveira

## Carregando XFiles

- A função `D3DXLoadMeshFromX` carrega o modelo armazenado em um arquivo XFILE e cria um objeto `ID3XMesh` a partir dele
- Arquivos XFILE contém informações sobre vértices, materiais e nomes dos arquivos de texturas
- Os arquivos de textura necessários para renderizar o modelo devem ser fornecidos separadamente

Copyright © Manuel M. Oliveira

# Carregando XFiles

- Exemplo

```
ID3DXMesh* Malha = NULL; // Armazena a malha lida do arquivo .X
std::vector<D3DMATERIAL9> Mtrls(0); // Armazena materiais lidos do arquivo .X
std::vector<IDirect3DTexture9*> Textures(0); // Armazena as texturas
ID3DXBuffer* adjBuffer = NULL; // Adjacência dos triangulos da malha
ID3DXBuffer* mtrlBuffer = NULL; // Array de materiais para a malha
DWORD numMtrls = 0; // Núm. de materiais diferentes da malha
...
bool Le_arquivo_XFile()
{
    ...
    if (FAILED( D3DXLoadMeshFromX("airplane 2.x", D3DXMESH_MANAGED,
        g_pd3dDevice, &adjBuffer, &mtrlBuffer, 0, &numMtrls, &Malha))
    {
        ::MessageBox(NULL, "D3DXLoadMeshFromX() – Falhou", "Erro", MB_OK);
        return false
    }
    ...
}
```

# Extraindo Materiais e Texturas de Arquivos XFiles

```
void Extrai_Materiais_e_Texturas()
{
    IDirect3DTexture9* tex = NULL; // ponteiro para objeto textura
    if (mtrlBuffer != NULL && numMtrls != 0 ) {
        D3DMATERIAL* mtrls = (D3DMATERIAL*) mtrlBuffer->GetBufferPointer();
        for (int i = 0; i < numMtrls; i++) {
            mtrls[i].MatD3D.Ambient = mtrls[i].MatD3D.Diffuse;
            Mtrls.push_back( mtrls[i].MatD3D ); // salva o conteúdo do i-ésimo material
            if (mtrls[i].pTextureFilename != NULL ) { // verifica se há textura associada
                D3DXCreateTextureFromFile(g_pd3dDevice, mtrls[i].pTextureFilename, &tex);
                Textures.push_back( tex ); // salva a textura carregada
            }
            else // o i-ésimo material não tem textura
                Textures.push_back( 0 );
        }
    }
    mtrlBuffer->Release(); // libera o buffer de materiais
}
```

Copyright © Manuel M. Oliveira

## Exibindo Modelos de XFiles

```
bool Display()
{
    ...
    g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0,0,0) |
        D3DCLEAR_ZBUFFER, 1.0f, 0 );

    g_pd3dDevice->BeginScene();

    for (int i = 0; i < Mtrlr.size(); i++) {           // para cada material
        g_pd3dDevice->SetMaterial( &Mtrlr[i] );       // seleciona o material com corrente
        g_pd3dDevice->SetTexture(0, Textures[i]);     // seleciona a textura associada
        Malha->DrawSubset(i);                         // renderiza a porção da malha associada ao material
    }

    g_pd3dDevice->EndScene();
    g_pd3dDevice->Present( NULL, NULL, NULL, NULL );
    return true;
}
```

Copyright © Manuel M. Oliveira

## Exercícios (8)



- Abra o projeto **prog\_08** (Carrega e exhibe arquivos XFile)
- Modifique o programa Prog\_08.cpp para permitir a **translação** dos objetos da cena nas direções X, Y e Z (SRU), usando **interação via mouse**
  - Utilize o botão da esquerda para fazer translação em X e Y
  - Utilize o botão da esquerda para fazer translação em Z
  - Durante a translação, o objeto deve manter sua orientação no momento em que a operação de translação teve início
  - Utilize as teclas "l" e "i" para reinicializar a posição do objeto em (0,0,0) no SRU.

Copyright © Manuel M. Oliveira

## Direct3D 9.0 - Conclusão

- API para criação de aplicações gráficas 3D, sendo parte integrante da DirectX 9.0
- Fácil de programar
- Suporta todo o pipeline gráfico: transformações, iluminação, mapeamento de textura, renderização de superfícies semi-transparentes, entre outras operações
- Oferece objetos pre-definidos, suporta *XFiles* e *Progressive Meshes*
- Oferece suporte para Microsoft *High-Level Shading Language* - HLSL

Copyright © Manuel M. Oliveira