Subject: Re: [Showoff] PIC
Posted by Sir Kane on Sat, 13 Dec 2008 22:56:45 GMT
View Forum Message <> Reply to Message

Omgomg

```
#include "dllmain.h"
#include "RegistrationSubsystems.h"
#include "Render.h"
#include "W32Font.h"
#include "SceneRender.h"
#include "RedirectHandler.h"
#include "Math.h"
#include <time.h>
#include "RenderObjClass.h"
#include "SceneClass.h"
#include "Console.h"
#include "StateManager.h"
#include "WeaponClasses.h"
#include "PICHUD.h"

CWin32Font* g_pPICFont;
CFontBatch* g_pPICBatch;

#define PICHUDSTATE_READY  (0)
#define PICHUDSTATE_RELOADING (1)
#define PICHUDSTATE_DEPLETED (2)

Vector3 g_DisplayVertices[5] = {
 Vector3(-0.008227f, 0.140099f, 0.024404f),
 Vector3(-0.056067f, 0.140099f, 0.024404f),
 Vector3(-0.059560f, 0.140099f, 0.040857f),
 Vector3(-0.059560f, 0.140099f, 0.028945f),
 Vector3(-0.008227f, 0.140099f, 0.040857f),
};

int g_DisplayIndices[9] = {
 3, 2, 1,
 1, 2, 4,
 1, 4, 0,
};

D3DCOLOR g_DisplayColors[5] = {
 D3DCOLOR_XRGB(0,0,0),
 D3DCOLOR_XRGB(0,0,0),
 D3DCOLOR_XRGB(0,0,0),
 D3DCOLOR_XRGB(0,0,0),
 D3DCOLOR_XRGB(0,0,0),
};
```

```
Vector3 g_OutsideVertices[9] = {
 Vector3(-0.002435f, 0.140099f, 0.024469f),
 Vector3(0.030383f, 0.140099f, 0.024469f),
 Vector3(0.040079f, 0.140099f, 0.026707f),
 Vector3(0.053505f, 0.140099f, 0.042370f),
 Vector3(0.054624f, 0.140099f, 0.049083f),
 Vector3(0.054624f, 0.140099f, 0.096073f),
 Vector3(0.035736f, 0.140099f, 0.131874f),
 Vector3(0.031261f, 0.140099f, 0.135230f),
 Vector3(0.003685f, 0.140099f, 0.135230f),
};

Vector3 g_InsideVertices[9] = {
 Vector3(-0.002435f, 0.140099f, 0.040133f),
 Vector3(0.027400f, 0.140099f, 0.040133f),
 Vector3(0.033367f, 0.140099f, 0.041251f),
 Vector3(0.038960f, 0.140099f, 0.047964f),
 Vector3(0.040079f, 0.140099f, 0.052439f),
 Vector3(0.040079f, 0.140099f, 0.093835f),
 Vector3(0.027641f, 0.140099f, 0.117330f),
 Vector3(0.024285f, 0.140099f, 0.119567f),
 Vector3(0.003685f, 0.140099f, 0.119567f),
};

float g_OutLengths[8];
float g_InLengths[8];

float g_OutLengthTotal;
float g_InLengthTotal;

#define VERT_Y_INC (0.000001f)

void InitNumDisplay(){
 int i;
 for (i = 0; i < 5; i++)
  g_DisplayVertices[i].Y += VERT_Y_INC;

}

void MakeBarLengths(){
 int i;

 for (i = 0; i < 9; i++)
  g_OutsideVertices[i].Y += VERT_Y_INC;

 for (i = 0; i < 9; i++)
  g_InsideVertices[i].Y += VERT_Y_INC;
```

```
 for (i = 0; i < 8; i++)
  g_OutLengthTotal += (g_OutLengths[i] =
(g_OutsideVertices[i+1]-g_OutsideVertices[i]).Length());

 for (i = 0; i < 8; i++)
  g_InLengthTotal += (g_InLengths[i] = (g_InsideVertices[i+1]-g_InsideVertices[i]).Length());
}

void DrawPICBar(float len, D3DCOLOR color, D3DCOLOR color2){
 float inlen;
 float alpha, left;
 int i, rects;
 Vector3 verts[4];
 D3DCOLOR colors[4];
 len = (len < 0.0f) ? 0.0f : ((len > 1.0f) ? 1.0f : len);
 inlen = g_InLengthTotal * len;
 int indices[6];

 left = inlen;
 i = 0;

 rects = len > 0.0f ? 1 : 0;

 for (i = 0; i < 8; i++){
  if (left < g_InLengths[i]) break;
  left -= g_InLengths[i];
  rects++;
 }

 left = inlen;

 indices[0] = 0;
 indices[1] = 1;
 indices[2] = 2;

 indices[3] = 1;
 indices[4] = 2;
 indices[5] = 3;

 colors[0] = color;
 colors[1] = color;
 colors[2] = color;
 colors[3] = color;

 for (i = 0; i < rects; i++){
  alpha = (left > g_InLengths[i]) ? 1.0f : 1.0f/g_InLengths[i]*left;
  left -= g_InLengths[i];
```

```
  verts[0] = g_InsideVertices[i];
  verts[1] = g_OutsideVertices[i];
  verts[2] = Vector3::Lerp(g_InsideVertices[i], g_InsideVertices[i+1], alpha);
  verts[3] = Vector3::Lerp(g_OutsideVertices[i], g_OutsideVertices[i+1], alpha);

  Draw_Indexed_Prim(2, verts, NULL, colors, indices);
 }

 left = g_InLengthTotal-inlen;

 colors[0] = color2;
 colors[1] = color2;
 colors[2] = color2;
 colors[3] = color2;

 for (i = 8; i > rects-1; i--){
  alpha = (left > g_InLengths[i-1]) ? 1.0f : 1.0f/g_InLengths[i-1]*left;
  left -= g_InLengths[i-1];

  verts[0] = g_InsideVertices[i];
  verts[1] = g_OutsideVertices[i];
  verts[2] = Vector3::Lerp(g_InsideVertices[i], g_InsideVertices[i-1], alpha);
  verts[3] = Vector3::Lerp(g_OutsideVertices[i], g_OutsideVertices[i-1], alpha);

  Draw_Indexed_Prim(2, verts, NULL, colors, indices);
 }
}

void DrawAmmoCountBackground(D3DCOLOR color){
 D3DCOLOR colors[9];
 int i;

 colors[0] = colors[1] = colors[2] = colors[3] = colors[4] = color;


 Draw_Indexed_Prim(3, g_DisplayVertices, NULL, colors, g_DisplayIndices);
}

void DrawChargeBar(float x, float y, float width, float height, float progress, D3DCOLOR
framecolor, D3DCOLOR barcolor, D3DCOLOR barcolor2){
 Vector3 verts[4];
 D3DCOLOR colors[4];
 int i;
 int indices[6] = {
  0, 1, 2,
  1, 2, 3,
 };
```

```
for (i = 0; i < 4; i++)
 verts[i].Z = 0.0f;

if (progress == 1.0f){
 for (i = 0; i < 4; i++)
  colors[i] = barcolor;

 verts[0].X = x;
 verts[0].Y = y;

 verts[1].X = x+width;
 verts[1].Y = y;

 verts[2].X = x;
 verts[2].Y = y+height;

 verts[3].X = x+width;
 verts[3].Y = y+height;
 Draw_Indexed_Prim(2, verts, NULL, colors, indices);
} else if (progress == 0.0f){
 for (i = 0; i < 4; i++)
  colors[i] = barcolor2;

 verts[0].X = x;
 verts[0].Y = y;

 verts[1].X = x+width;
 verts[1].Y = y;

 verts[2].X = x;
 verts[2].Y = y+height;

 verts[3].X = x+width;
 verts[3].Y = y+height;
 Draw_Indexed_Prim(2, verts, NULL, colors, indices);
} else {
 for (i = 0; i < 4; i++)
  colors[i] = barcolor;

 verts[0].X = x;
 verts[0].Y = y;

 verts[1].X = x+(width*progress);
 verts[1].Y = y;

 verts[2].X = x;
 verts[2].Y = y+height;
```

```
  verts[3].X = x+(width*progress);
  verts[3].Y = y+height;
  Draw_Indexed_Prim(2, verts, NULL, colors, indices);


  for (i = 0; i < 4; i++)
   colors[i] = barcolor2;

  verts[0].X = x+(width*progress);
  verts[0].Y = y;

  verts[1].X = x+width;
  verts[1].Y = y;

  verts[2].X = x+(width*progress);
  verts[2].Y = y+height;

  verts[3].X = x+width;
  verts[3].Y = y+height;
  Draw_Indexed_Prim(2, verts, NULL, colors, indices);
 }

 Draw_Frame(RectClass(x, y, x+width, y+height), framecolor);
}

/*
PICHUDRenderObjClass
*/

PICHUDRenderObjClass::PICHUDRenderObjClass(){
 Matrix3D mat(true);
 Set_Transform(mat);
 m_Bits |= BOUNDING_VOLUMES_VALID | IS_FORCE_VISIBLE;

}

PICHUDRenderObjClass::~PICHUDRenderObjClass(){
}

RenderObjClass* PICHUDRenderObjClass::Clone(){
 return NULL;
}

int PICHUDRenderObjClass::Get_Sort_Level(){
 return WW3D::MinStaticSortLevel;
 //return WW3D::MaxStaticSortLevel;
 ShaderClass shader;
 shader.Set_Shader(ShaderClass::_PresetAlphaShader);
```

```
 shader.m_Depth_Mask = 1;
 return shader.Guess_Sort_Level();
}

void PICHUDRenderObjClass::Render(RenderInfoClass&){
 int sort_level;
 CTextDrawer draw;
 int _ammo;
 int maxammo;
 int displayammo;
 Matrix3D mat, trans;
 WeaponClass *pWeapon;
 Matrix4 mat2;
 float ammo;
 ShaderClass shader;
 wchar_t lBuf[16];
 bool hasinclip;
 float statetime;
 float reloadtime;
 WeaponClass::WeaponState state;
 float barx;
 float time;
 int t;

 if (WW3D::AreStaticSortListsEnabled && (sort_level = Get_Sort_Level())){
  WW3D::Add_To_Static_Sort_List(this, sort_level);
  return;
 }

 g_pPICBatch->Reset();
 draw.Set_Batch(g_pPICBatch);
 draw.Set_Font(g_pPICFont);


 D3DMATERIAL9 matr;
 memset(&matr, 0, sizeof(D3DMATERIAL9));
 matr.Diffuse.a = 1.0f;
 matr.Diffuse.r = 1.0f;
 matr.Diffuse.g = 1.0f;
 matr.Diffuse.b = 1.0f;

 (*g_ppD3DDevice)->SetPixelShader(NULL);
 (*g_ppD3DDevice)->SetVertexShader(NULL);
 StateManager::SetRenderState(D3DRS_DIFFUSEMATERIALSOURCE, D3DMCS_COLOR1);
 StateManager::SetMaterial(&matr);
 StateManager::SetRenderState(D3DRS_LIGHTING, FALSE);
 StateManager::SetRenderState(D3DRS_FOGENABLE, FALSE);
```

```
 StateManager::SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);

 shader.Set_Shader(ShaderClass::_PresetAlphaShader);
 shader.m_Texturing = 1;
 shader.Apply();


 (*g_ppD3DDevice)->SetTexture(0, NULL);
 mat2.Init(m_Transform);
 mat2 = (Matrix4(m_Transform).Transpose());
 StateManager::SetTransform(D3DTS_WORLD, (D3DMATRIX*)&mat2);
 StateManager::SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);
 displayammo = -1;
 hasinclip = false;
 state = WeaponClass::WS_READY;
 statetime = 0;
 reloadtime = 1.0f;
 if ((pWeapon = WeaponBagClass::GetMyWeapon(false))){
  state = pWeapon->GetState();
  statetime = pWeapon->GetStateTime();
  reloadtime = pWeapon->m_pDefintion->m_Reload_Time;
  if (reloadtime == 0.0f) reloadtime = 0.0000001f;
  _ammo = pWeapon->m_Ammo;
  maxammo = pWeapon->m_Max_Ammo;
  if (_ammo == -1 || maxammo == -1){
   ammo = 1.0f;
   displayammo = -1;
  }else{
   ammo =
1.0f/((float)((int)pWeapon->m_pDefintion->m_Max_Inventory_Rounds))*((float)(displayammo =
maxammo));
   hasinclip = (int)pWeapon->m_Ammo ? true : false;
  }
 } else
  ammo = 1.0f;
 //Draw ammo bar
 DrawPICBar(ammo, D3DCOLOR_XRGB(0,255,0), D3DCOLOR_XRGB(255,0,0));
 DrawAmmoCountBackground(D3DCOLOR_XRGB(0,0,0));

 //Draw weapon status
 draw.Set_Scene_Size(0.01f);
 mat.Make_Identity();
 mat.setRotationX(Deg2Rad(-90.0f));
 mat.setRotationZ(Deg2Rad(180.0f));
 trans.Set_Translation(Vector3(0.023128f-0.009f, 0.1402f, 0.051910f+0.01f));
 mat = m_Transform*trans*mat;
 mat2 = Matrix4(mat).Transpose();
 StateManager::SetTransform(D3DTS_WORLD, (D3DMATRIX*)&mat2);
```

```
 if (state == WeaponClass::WS_RELOADING){
  barx = (float)draw.Draw_Single_Line(D3DCOLOR_XRGB(255,255,0), 0, 0, L"Charging... ");
  barx *= draw.GetScalar();
  DrawChargeBar(barx, 0.003f, 0.02f, 0.004f, 1.0f-(1.0f/reloadtime*statetime),
D3DCOLOR_XRGB(255,255,0), D3DCOLOR_XRGB(255,255,0), D3DCOLOR_XRGB(0,0,0));
 } else if (maxammo == 0 && _ammo == 0)
  draw.Draw_Single_Line(D3DCOLOR_XRGB(255,0,0), 0, 0, L"Battery depleted.");
 else
  draw.Draw_Single_Line(D3DCOLOR_XRGB(0,255,0), 0, 0, L"Charged.");



 if (g_pPICBatch->Serialize())
  g_pPICBatch->Render();
 g_pPICBatch->Reset();

 //DrawAmmoCountBackground(hasinclip ? D3DCOLOR_XRGB(0,255,0) :
D3DCOLOR_XRGB(255,0,0));

 draw.Set_Scene_Size(0.0169f);
 mat.Make_Identity();
 mat.setRotationX(Deg2Rad(-90.0f));
 mat.setRotationZ(Deg2Rad(180.0f));
 trans.Set_Translation(Vector3(-0.008227f+0.0001f, 0.1402f, 0.024404f+0.0162f));
 mat = m_Transform*trans*mat;
 mat2 = Matrix4(mat).Transpose();
 StateManager::SetTransform(D3DTS_WORLD, (D3DMATRIX*)&mat2);

 if (displayammo == -1)
  StrCpyW(lBuf, L"\x221E");
 else
  swprintf(lBuf, L"%d", displayammo);


 draw.Draw_Single_Line(displayammo > 0 ? D3DCOLOR_XRGB(0,255,0) :
D3DCOLOR_XRGB(255,0,0), 0, 0, lBuf);
 if (g_pPICBatch->Serialize())
  g_pPICBatch->Render();
}

RenderObjClass*  _stdcall AttachWeapon(WeaponClass* pWeapon, RenderObjClass*
pWeaponObj){
 g_pWeaponModel = pWeaponObj;
 RenderObjClass *pObj;
 if (pWeapon->m_pDefintion->Get_ID() == 409610033){
  pObj = new PICHUDRenderObjClass;
  if (pObj && pWeaponObj){
```

```
   pWeaponObj->Add_Sub_Object_To_Bone(pObj, "F_GM_SCOPE");
   pObj->Release_Ref();
  }
 }
 return pWeaponObj;
}

__declspec(naked) void ASM_WeaponAttach(){
 __asm {
  add esp, 4;
  push eax;
  push edi;
  call AttachWeapon;
  mov edx, 0x0070E7FC;
  jmp edx;
 }
}

/*
PIDHUDRegistrator
*/

void PIDHUDRegistrator::Init(){
#ifdef GAME
 g_pRedirectHandler->RedirectCall(0x0070E7F4, ASM_WeaponAttach);
 g_pPICFont = new CWin32Font();
 g_pPICFont->Create("Arial", 32, 1, 0, 0, 0);
 g_pPICBatch = new CFontBatch;
 InitNumDisplay();
 MakeBarLengths();

#endif //GAME
}


void PIDHUDRegistrator::Terminate(){
#ifdef GAME
 delete g_pPICFont;
 delete g_pPICBatch;
#endif //GAME
}

PIDHUDRegistrator g_PIDHUDRegistrator;
```