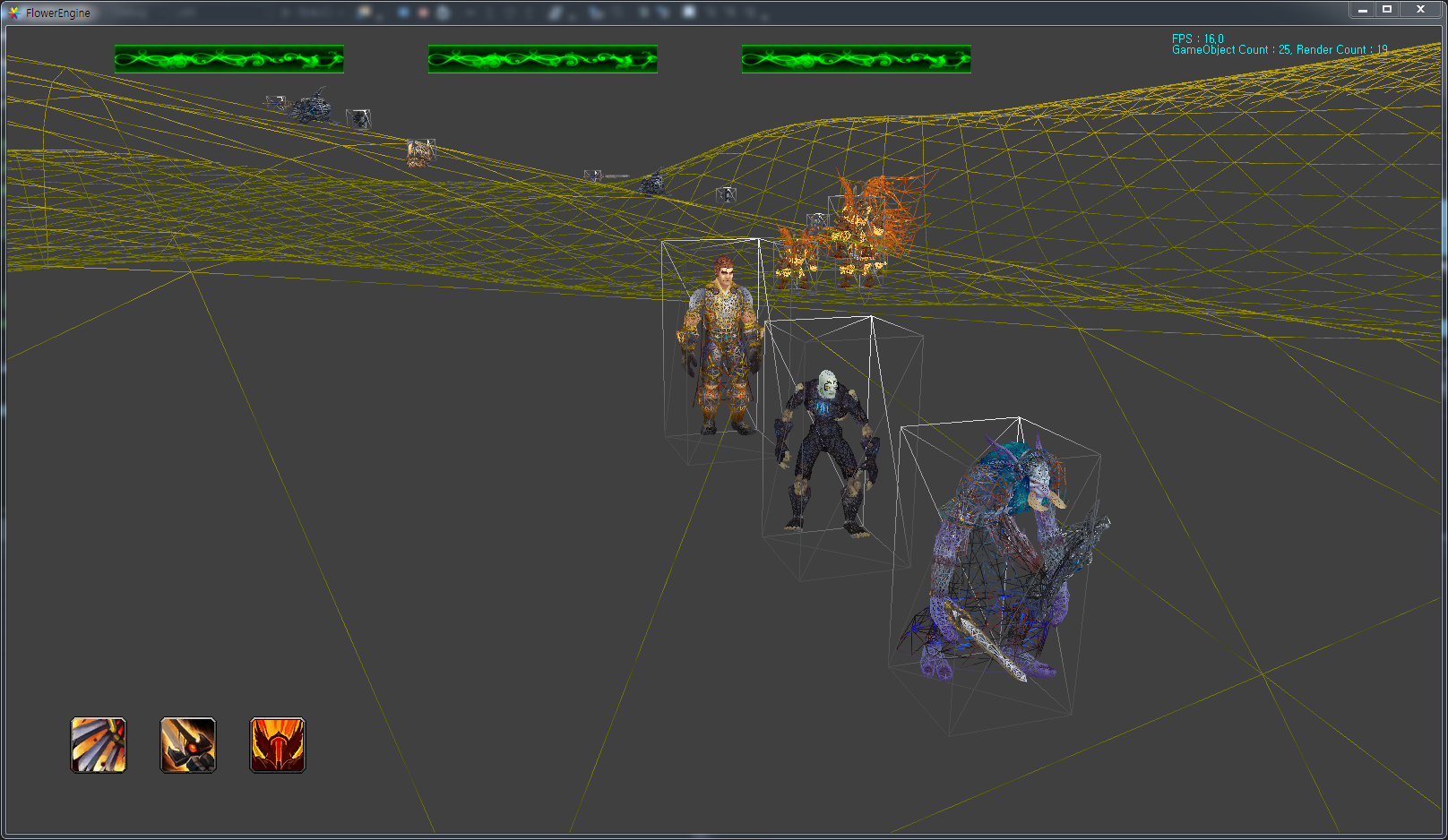
**포트폴리오**

**클라이언트 프로그래머**



**작성자 : 송현국**

[hkn10004@naver.com](mailto:hkn10004@naver.com)

HP : 010-3592-5921

팀원 : 김동오, 이호형

선생님 : 서울게임아카데미(SGA) 김주안

목차

1장 : 전체적인 설명

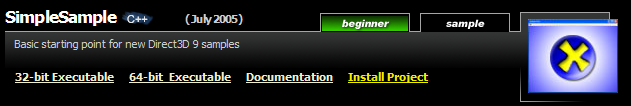
2장 : 전투 개발

3장 : 전투 외 개발

게임 개발의 시작

저는 경력 5년이상 된 프로그래머입니다. 개인적으로 공부를 더 하고 안정적인 개발자가 되기 위해서 국비지원 수업을 찾던 중 서울게임아카데미의 수업과정을 보고 신청하게 되었고 DirectX API를 이용하여 개발하게 되었습니다. 엔진은 언리얼, 유니티, Cocos2D-X 어떠한 엔진을 써도 상관은 없으나 함께 공부하는데에 DirectX를 사용하므로 자체엔진을 개발하게 되었습니다.

기반 윈도우 프레임워크



일단, DXUT를 사용하면 DirectX Sample에서 제공하는 UI등 많은 것들을 사용할 수 있으므로 DirectX Samples에서 제공하는 SimpleSample을 사용하였습니다. 물론, 이 예제들에서 제공하는 것들을 가공해야 합니다.

프로그램의 시작 WinMain

콜백함수란 함수 포인터로 프레임워크에 필요한 함수를 재정의 해서 그 함수를 연결하는 것입니다. DXUT에서 필요한 부분에 연결한 함수가 호출됩니다.

콜백함수를 간단히 설명하자면 윈도우 메시지 처리, 키보드 마우스 입력 처리, 프레임 업데이트, 렌더링, 디바이스 세팅 관련 콜백함수들이 있습니다.

|  |
| --- |
| //--------------------------------------------------------------------------------------  // Entry point to the program. Initializes everything and goes into a message processing  // loop. Idle time is used to render the scene.  //--------------------------------------------------------------------------------------  int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow)  {  // Set DXUT callbacks  DXUTSetCallbackMsgProc(MsgProc);  DXUTSetCallbackKeyboard(OnKeyboard);  DXUTSetCallbackFrameMove(OnFrameMove);  DXUTSetCallbackDeviceChanging(ModifyDeviceSettings);  DXUTSetCallbackD3D9DeviceAcceptable(IsD3D9DeviceAcceptable);  DXUTSetCallbackD3D9DeviceCreated(OnD3D9CreateDevice);  DXUTSetCallbackD3D9DeviceReset(OnD3D9ResetDevice);  DXUTSetCallbackD3D9DeviceLost(OnD3D9LostDevice);  DXUTSetCallbackD3D9DeviceDestroyed(OnD3D9DestroyDevice);  DXUTSetCallbackD3D9FrameRender(OnD3D9FrameRender);  InitApp();  DXUTInit(true, true, NULL);  DXUTSetCursorSettings(true, true);  DXUTCreateWindow(L"FlowerEngine");  DXUTCreateDevice(true, 1600, 900);    Camera::GetInstance()->Init();  SceneManager::GetInstance()->AddScene(new SceneRPG("SceneRPG"));  /\*SceneManager::GetInstance()->AddScene(new SceneAircraft("SceneAircraft"));  SceneManager::GetInstance()->AddScene(new ScenePicking("ScenePicking"));\*/  SceneManager::GetInstance()->ChangeScene("SceneRPG");  SceneManager::GetInstance()->Awake();    DXUTMainLoop(); // Enter into the DXUT render loop  return DXUTGetExitCode();  } |

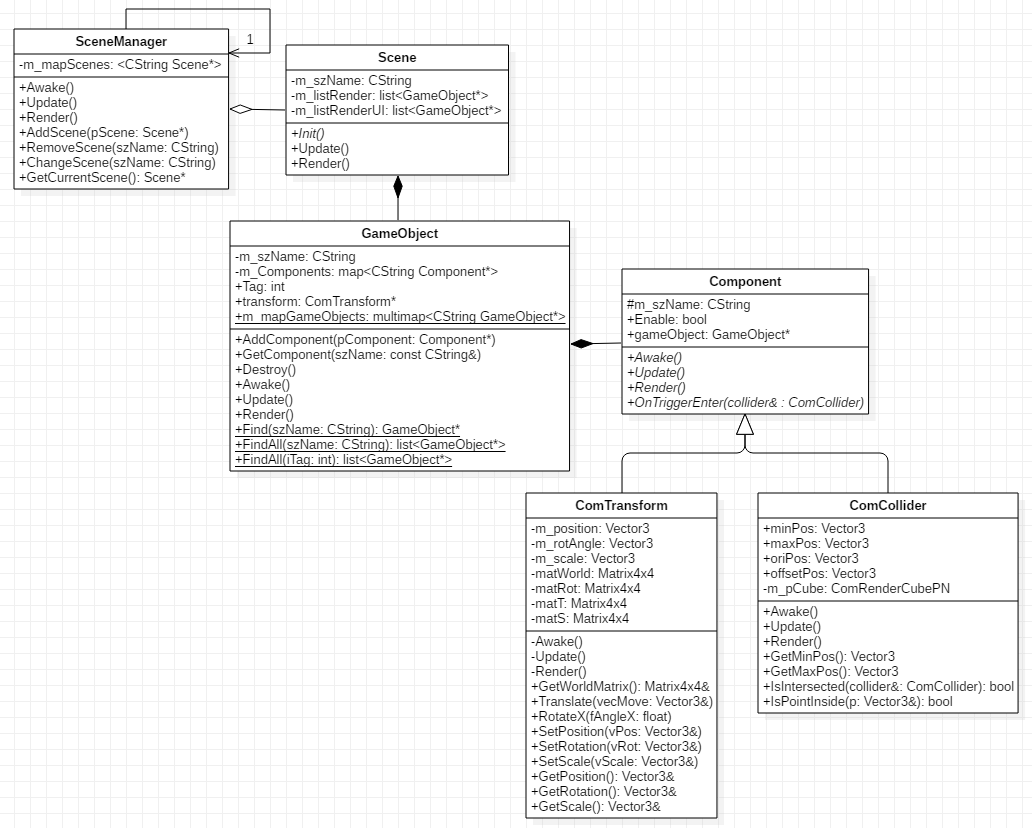
GameObject Component System 설계

많은 분께서 아시다시피 GameProgramming Gems 6권에도 소개되어 있고 유니티, 언리얼등에서 사용하는 설계방식 입니다. 기본 내용을 공부하여 코드 작성 하였습니다.

간략히 설명 드리겠습니다. 위 코드에 보면 SceneManager를 사용하여 시작하는 것을 볼 수 있습니다. SceneManager는 Scene 관리를 담당하는 싱글턴 클래스입니다. 씬을 추가하거나 뺄 수 있고 변경할 수 있습니다. 그리고 현재 씬 객체를 가져오는 함수가 있습니다. 씬에는 추가한 GameObject 객체들이 있고 GameObject에는 구성요소(Component)들이 포함되어 있습니다.

Component의 하위 클래스인 ComTransform은 게임오브젝트의 위치, 회전, 크기 변환을 담당합니다. ComCollider는 큐브의 좌하단 우상단 정점을 이용한 AABB 충돌검사 방식입니다.

그림(UML)으로 설명하자면 다음과 같습니다.



프로그래밍 언어.

C언어 기본기에 대해서 설명 드리겠습니다. 포인터(\*)를 사용하는 이유는 다양한 이유가 있지만 인자값으로 객체를 넘기면 그 메모리 크기만큼 복사가 일어납니다. 그러므로 4byte 주소값인 포인터를 사용합니다. 또한, 배열을 사용할 때도 포인터를 사용하여 크기만큼 메모리를 할당하고 사용하고 해제합니다. 메모리의 어떠한 값에 접근하고자 할 때에도 포인터를 사용하여 그 주소값을 얻어오고 그 주소값을 통하여 메모리에 할당된 값에 접근하여 그 값을 사용합니다. 이중 포인터(\*\*)는 포인터의 배열을 만들 때 사용합니다.

레퍼런스(&)는 그 객체의 다른 이름입니다. 선언과 동시에 초기화 해주어야 합니다. 레퍼런스를 사용하면 그 객체만큼 메모리가 복사되는 것이 아닌 레퍼런스의 크기만큼만 사용됩니다.

C++의 기본기에 대해서 설명 드리겠습니다. C++은 C와는 달리 객체지향 언어입니다. Java와 C#도 객체지향 언어입니다. 객체지향 언어의 특징은 프로그램에서 하나의 개념 또는 사물을 하나의 클래스로 구현한 것입니다. 그 클래스는 매니저가 될 수도 있고 인물, 자동차, 과일과 같은 것이 될 수도 있습니다. 객체지향 개념은 C++, Java, C#이 동일합니다.

여기에서 정보의 은닉성 개념이 있습니다. private, protected, public인데 클래스에서 변수를 사용할 때, 자기 자신만 사용하는가(private), 자식 클래스도 사용하는가(protected), 외부 모든 곳에서 사용하는가(public)입니다. 위 그림에서 비공개(private)은 –로 표기, 자식 공개(protected)는 #으로 표기, 모두 공개(public)은 +로 표기합니다.

상속 개념이 있습니다. 즉 기반 클래스(부모)와 상속된 클래스(자식)으로 나눌 수 있는데 위에서는 Component의 자식 클래스가 ComTransform이고 자식 클래스는 부모 클래스의 protected로 선언된 변수들을 사용할 수 있습니다. 함수도 마찬가지 입니다. 가상 함수(virtual)로 선언된 함수를 재정의(override)해서 사용하면 컴파일러는 재정의된 함수를 호출합니다. 순수 가상 함수는 필수적으로 재정의 해서 사용해야 할 함수가 있을 때 사용합니다. 상속관계는 위 그림에서 삼각형으로 표기합니다.

static 변수는 선언하면 메모리에 그 변수 하나만 존재합니다. 싱글턴 클래스는 이 static의 성질을 이용한 것입니다. static 함수는 외부에서 그 클래스의 함수로 바로 접근하여 사용할 수 있게합니다. static 변수는 프로그램이 종료되는 시점에 메모리해제 됩니다. 위 그림(UML)에서 밑줄 그어진 함수가 static 함수이며 이탤릭체로 표기된 함수가 가상함수입니다.

렌더링 파이프라인

RGB 픽셀로 표시할 수 있는 전자기기에 그 픽셀을 그리기(Rendering) 위해서 계산하는 과정을 말합니다. 가장 기본 적으로 정점(Vertex) 3개로 렌더링을 해 줄 수 있습니다. 이 정점들을 월드 변환, 뷰 변환, 투영 변환을 해주어야 합니다. 그래픽 관련 계산을 빠르게 하기 위해서 그래픽 처리 장치(Graphics Processing Unit)를 사용하는데 이를 정점 셰이더 픽셀 셰이더라 합니다.

월드 변환

각 정점은 벡터로 정의합니다. V1(0, 1, 0) V2(1, 0, 0), V3(-1, 0, 0) 이렇게 3개의 정점을 선언하면 이것은 로컬 공간의 위치가됩니다. 이 정점들을 월드 공간으로 보내주기 위해서 행렬(Matrix4x4)을 사용합니다.

행렬에는 크기 변환 행렬, 회전 행렬, 이동 행렬이 있습니다. 이 행렬을 모두 곱한값이 월드 행렬(World Matrix)이 됩니다.

|  |
| --- |
| pDevice9->SetTransform(D3DTS\_WORLD, &gameObject->transform->GetWorldMatrix()); |

코드 설명 : 디바이스를 통해 변환을 설정( 월드 변환, &월드 행렬 ); 주로 렌더링 하기 직전에 설정해 줍니다.

뷰 변환

월드 공간에 배치된 오브젝트들을 Z축을 바라보는 카메라 기준으로 변경하는 변환을 말합니다. 뷰 행렬이 있습니다.

|  |
| --- |
| D3DXMatrixLookAtLH(&m\_matView, &m\_eye, &m\_lookat, &m\_up);  pDevice9->SetTransform(D3DTS\_VIEW, &m\_matView); |

코드 설명 : 카메라의 위치(m\_eye), 바라 보는 곳(m\_lookat), 위 방향(m\_up) 세개의 벡터를 사용하여 뷰 행렬을 만들고 디바이스를 통해 변환을 설정( 뷰 변환, &뷰 행렬 ); 카메라 Update() 부분에서 설정해 줍니다.

투영 변환

X, Y축은 -1~1사이 Z축은 0~1사이로 변환됩니다. 이 과정에서 카메라에 멀리 있는 정점들은 작게 보이게 됩니다. 카메라가 설정될 때 설정해 줍니다.

|  |
| --- |
| RECT clientRect = DXUTGetWindowClientRect();  // 투영 행렬 왼손 좌표계, 시야각 45도, 화면 사이즈 종횡비, 보이는 거리 1~1000  D3DXMatrixPerspectiveFovLH(&m\_matProj, D3DX\_PI / 4.0f, clientRect.right / (float)clientRect.bottom, 1, fDistZ);  pDevice9->SetTransform(D3DTS\_PROJECTION, &m\_matProj); |

셰이더

셰이더를 사용하는 방법은 DirectX와 유니티, 언리얼 엔진등 각각 다릅니다만 근본적인 내용은 같습니다.

DirectX에서 사용방법 : m\_pEffect를 통해 HLSL의 변수에 접근합니다.

|  |
| --- |
| typedef LPD3DXEFFECT Shader;  Shader m\_pEffect;  D3DXCreateEffectFromFile(pDevice9, szFilepath, NULL, NULL, D3DXSHADER\_DEBUG, NULL, &m\_pEffect, &pError); |

HLSL : 가장 기본적인 정점 셰이더 월드, 뷰, 투영 변환과 픽셀 셰이더 텍스쳐 UV 적용 셰이더 코드입니다.

|  |
| --- |
| // 행렬 선언  float4x4 gWorldMatrix : World;  float4x4 gViewMatrix : View;  float4x4 gProjMatrix : Projection;  // 텍스쳐 선언  texture DiffuseMap\_Tex  sampler2D DiffuseSampler = sampler\_state  {  Texture = (DiffuseMap\_Tex);  MipFilter = LINEAR; // 픽셸들을 선형보간하여 부드럽게 표현  MinFilter = LINEAR;  MagFilter = LINEAR;  };  // 입력받는 정점 정보  struct VS\_INPUT  {  float4 Position : POSITION;  float2 TexCoord : TEXCOORD0;  };  // 계산되어 출력되는 정점 정보  struct VS\_OUTPUT  {  float4 Position : POSITION;  float2 TexCoord : TEXCOORD0;  };  // 정점 셰이더 함수  VS\_OUTPUT Vertex\_Shader\_main(VS\_INPUT Input)  {  VS\_OUTPUT Output;  Output.Position = mul(Input.Position, gWorldMatrix); // 정점들을 월드 공간으로  Output.Position = mul(Output.Position, gViewMatrix); // 정점들을 뷰 공간으로  Output.Position = mul(Output.Position, gProjMatrix); // 정점들을 투영 공간으로  Output.TexCoord = Input.TexCoord;    return Output;  }  // 입력 받는 픽셸 정보  struct PS\_INPUT  {  float2 TexCoord : TEXCOORD0;  };  // 픽셀 셰이더 함수  float4 Pixel\_Shader\_main(PS\_INPUT Input) : COLOR  {  float3 color = tex2D(DiffuseSampler, Input.TexCoord).rgb; // Texture에서 UV값을 가져옴  return float4(color, 1);  } |

텍스쳐 UV란 U(1, 0) V(0, 1)입니다. 32x32 텍스쳐를 사용할 때 U(0.5)값은 16번째 텍셀값입니다.

비례식을 사용해서 텍셀을 구하려면 (32x32 텍스쳐, U(0.5)일 때)

x : 32 = 0.5 : 1

x = 32 \* 0.5 = **16**

비례식을 사용해서 U를 구하려면

16 : 32 = x : 1

32 \* x = 16

x = 16 / 32 = **0.5**