НАЦИОНАЛЬНЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ

«КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ»

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ

ЛАБОРАТОРНАЯ РАБОТА **№4**

с курса

«Компьютерная графика»

**Выполнила :**

студентка 3 курса

ФПМ

КВ-81

Удругова Настя

**Проверила:**

Потапова К.Р.

Киев 2011

Описание и алгоритм:

В этой лабараторной работе я продемонстрировала моделирование ткани на GPU, с помощью DirectX10.

При запуске приложения Вы видите сферу в центре помещения и рядом висящую ткань. В качестве текстур используются .dds файлы. Вы можете задать некоторые характристики для моделей с помощью панели управления:

напрвление ветра, сила ветра, свойства ткани.

Ткань можно перемещать с помощью левой кнопки мыши, изменить угол обзора сцены – с помощью правой кнопки мыши;

Ткань можно разрезать. Для этого нужно возпользоваться: Левая кнопка мыши + CTRL; С помощью клавиши Z можно востонавить целосность ткани. А для того, чтоб отменить все действия, и вернуть программу на начальный уровень нужно воспользовать клавишей 0.

Ткань моделируется как множество частиц. Каждая частица подвергается:

1. Внешним силам, таким как ветер, гравитация, перемещение;

2. Различным ограничениям, таким как:

а) поддержание общей формы объекта (пружинное ограничение);

б) предотвращение взаимопроникновение с окружающей средой;

Уровнение движения частицы в результате применения внешних сил интегрирован использованием Велве интеграции. Различные ограничения создают систему уравнений, соединяющую позиции частиц.  Эта система решается на каждом шаге моделирования, путем применения ограничений один за другим для заданного числа итераций.

Пружинное ограничение между двумя частицами, P и Q, определено как ограничение расстояния между P и Q.

Среда определена как ряд объектов, а именно сферы и плоскости. Ограничение при столкновении объектов обеспечивается путем проверки, является ли частица внутри объекта или нет; и если она внутри, перемещаем позицию частицы на поверхность объекта, обычно в ту точку на поверхности объекта, которая есть самой близкой.

Алгоритм для каждого шага моделирования ткани:

**Шаг 1:**

Выбрать вершинные шейдеры, к которым применяется сила;

Установить позицию буфера (GPU);

Рендеринг списка точек;

Подкачка буферов позиций;

**Шаг 2:**

Установить пиксельный шейдер, который обновляет позицию;

Установить позицию буфера, как рендеринга;

Подкачка буферов позиций;

**Шаг 3:** Для каждого шага расслабления:

**Шаг 3а:** Для каждого множества независимых групп пружинных ограничений:

Установить геометрические шейдеры, к которым относится расстояние ограничений (SatisfyStructuralAndShearSpringConstraints, SatisfyShearSpringConstraints)

Установить позицию буфера, как поток выходных целей

Рендеринг как индексированный список строки со смежностью

Подкачка буферов позиций

**Шаг 3б:**

Для каждой частицы осуществление ограничения коллизии (**SatisfyCollisionConstraints**)

Установить позицию буфера, как поток выходных целей

Рендеринг как индексированный список точек;

Подкачка буферов позиций;

Код программы

Cloth.cpp

#include "DXUT.h"

#include "DXUTcamera.h"

#include "DXUTgui.h"

#include "DXUTsettingsdlg.h"

#include "DXUTShapes.h"

#include "SDKmesh.h"

#include "SDKmisc.h"

#include "resource.h"

#include "NVUTMedia.h"

#include "Cloth.h"

#include <stddef.h>

//--------------------------------------------------------------------------------------

// UI control IDs

//--------------------------------------------------------------------------------------

enum {

IDC\_TOGGLEFULLSCREEN,

IDC\_TOGGLEREF,

IDC\_CHANGEDEVICE,

IDC\_RUN\_PAUSE\_SIMULATION,

IDC\_STEP\_SIMULATION,

IDC\_NUM\_SIM\_PASSES\_LABEL,

IDC\_NUM\_SIM\_PASSES,

IDC\_WIND\_HEADING\_LABEL,

IDC\_WIND\_HEADING,

IDC\_WIND\_STRENGTH\_LABEL,

IDC\_WIND\_STRENGTH,

IDC\_UNCUT,

IDC\_RESET\_SIMULATION,

IDC\_SHOW\_TANGENT\_SPACE,

IDC\_WIREFRAME,

};

//--------------------------------------------------------------------------------------

// Global variables

//--------------------------------------------------------------------------------------

CModelViewerCamera g\_Camera; // A model viewing camera

CDXUTDialogResourceManager g\_DialogResourceManager; // manager for shared resources of dialogs

CD3DSettingsDlg g\_SettingsDlg; // Device settings dialog

CDXUTDialog g\_HUD; // manages the 3D UI

CDXUTDialog g\_SampleUI; // dialog for sample specific controls

ID3DX10Font\* g\_Font;

ID3DX10Sprite\* g\_Sprite;

ID3D10Effect\* g\_Effect;

CDXUTTextHelper\* g\_TxtHelper;

// Cloth size

const int g\_Width = 31;

const int g\_Height = 31;

int g\_NumVertices = g\_Width \* g\_Height;

const float g\_TexCoordScale = 4;

// Position indices

int g\_Old = 0;

int g\_Current = 1;

int g\_New = 2;

// Simulation

enum {

PASS\_SIMULATE\_FORCE,

PASS\_SIMULATE\_STRUCTURAL\_AND\_SHEAR\_SPRING\_0,

PASS\_SIMULATE\_STRUCTURAL\_AND\_SHEAR\_SPRING\_1,

PASS\_SIMULATE\_SHEAR\_SPRING\_0,

PASS\_SIMULATE\_SHEAR\_SPRING\_1,

PASS\_SIMULATE\_COLLISION

};

int g\_NumSimPasses = 3;

int g\_NumPrimitivesPerBatch = 4;

int g\_LastPrimitiveIndex = g\_NumVertices / g\_NumPrimitivesPerBatch;

int g\_LastPrimitiveSize = g\_NumVertices % g\_NumPrimitivesPerBatch;

int g\_NumSimIndices = g\_NumPrimitivesPerBatch \* (g\_LastPrimitiveIndex + (g\_LastPrimitiveSize == 0 ? 0 : 1));

#define NUM\_SIMULATE\_IB (PASS\_SIMULATE\_COLLISION + 1)

int g\_NumSimulatedPrimitives[NUM\_SIMULATE\_IB];

int g\_WindHeading;

int g\_WindStrength;

int g\_NumCutterTriangles;

int g\_FirstCutterTriangle;

// Rendering

int g\_NumIndices;

// Vertex buffers

ID3D10Buffer\* g\_InitialPositionVB;

ID3D10Buffer\* g\_TexCoordVB;

ID3D10Buffer\* g\_AnchorVB;

ID3D10Buffer\* g\_ParticleVB[3];

ID3D10Buffer\* g\_NormalTangentVB;

ID3D10Buffer\* g\_CutterEdgeVB;

// Constant buffers

ID3D10Buffer\* g\_CutterEdgeCB;

// Index buffers

ID3D10Buffer\* g\_SimulateIB[NUM\_SIMULATE\_IB];

ID3D10Buffer\* g\_NormalTangentIB;

ID3D10Buffer\* g\_RenderIB;

// Vertex layouts

ID3D10InputLayout\* g\_ApplyForcesIL;

ID3D10InputLayout\* g\_SatisfyConstraintsIL;

ID3D10InputLayout\* g\_RenderTangentSpaceIL;

ID3D10InputLayout\* g\_AnchorIL;

ID3D10InputLayout\* g\_RenderClothIL;

ID3D10InputLayout\* g\_RenderMeshIL;

// Render target view

ID3D10RenderTargetView\* g\_ParticleRTV[3];

ID3D10RenderTargetView\* g\_CutterEdgeRTV;

// Shader resource view

ID3D10ShaderResourceView\* g\_ParticleSRV[3];

// Render state

ID3D10RasterizerState\* g\_WireframeRS;

ID3D10RasterizerState\* g\_SolidRS;

// Anchor points

D3DXVECTOR2 g\_AnchorPoint[] = {

D3DXVECTOR2(0, 0),

D3DXVECTOR2(g\_Width / 2, 0),

D3DXVECTOR2(g\_Width - 1, 0)

};

int g\_NumAnchorPoints = sizeof(g\_AnchorPoint) / sizeof(g\_AnchorPoint[0]);

D3DXVECTOR3 g\_Center;

bool g\_TransformAnchorPoints;

D3DXMATRIX g\_Transform;

// Transforms

D3DXMATRIX g\_Projection;

D3DXMATRIX g\_View;

D3DXMATRIX g\_ViewProjection;

D3DXVECTOR3 g\_Eye;

// Scene

Plane g\_Plane[] = {

{ D3DXVECTOR3(0, 1, 0), 3 },

{ D3DXVECTOR3(0, -1, 0), 6 },

{ D3DXVECTOR3(1, 0, 0), 7 },

{ D3DXVECTOR3(-1, 0, 0), 7 },

{ D3DXVECTOR3(0, 0, 1), 7 },

{ D3DXVECTOR3(0, 0, -1), 7 }

};

const int g\_NumPlanes = sizeof(g\_Plane) / sizeof(g\_Plane[0]);

D3DXMATRIX g\_PlaneWorld[g\_NumPlanes];

ID3D10ShaderResourceView\* g\_PlaneDiffuseTexture[g\_NumPlanes];

ID3D10ShaderResourceView\* g\_PlaneNormalMap[g\_NumPlanes];

ID3DX10Mesh\* g\_PlaneMesh;

Sphere g\_Sphere[] = {

{ D3DXVECTOR3(0, 0, 0), 0.4f },

};

const int g\_NumSpheres = sizeof(g\_Sphere) / sizeof(g\_Sphere[0]);

D3DXMATRIX g\_SphereWorld[g\_NumSpheres];

ID3DX10Mesh\* g\_SphereMesh;

// Textures

ID3D10ShaderResourceView\* g\_CellDiffuseTexture[3];

ID3D10ShaderResourceView\* g\_CellNormalMap[3];

ID3D10ShaderResourceView\* g\_ClothDiffuseTexture;

ID3D10ShaderResourceView\* g\_ClothNormalMap;

// Commands

bool g\_Help = true;

bool g\_Reset;

bool g\_Run = true;

int g\_StepSimulation;

bool g\_Uncut;

bool g\_Wireframe;

bool g\_ShowTangentSpace;

bool g\_ShiftDown;

bool g\_CtrlDown;

// Backbuffer attributes

ID3D10RenderTargetView\* g\_BackBufferRTV;

ID3D10DepthStencilView\* g\_BackBufferDSV;

D3D10\_VIEWPORT g\_BackBufferVP;

int g\_BackBufferWidth;

int g\_BackBufferHeight;

//--------------------------------------------------------------------------------------

// Forward declarations

//--------------------------------------------------------------------------------------

bool CALLBACK ModifyDeviceSettings(DXUTDeviceSettings\* pDeviceSettings, void\* pUserContext);

void CALLBACK OnFrameMove(double time\_d, float elapsedTime, void\* pUserContext);

LRESULT CALLBACK MsgProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam, bool\* pbNoFurtherProcessing, void\* pUserContext);

void CALLBACK OnKeyboard(UINT nChar, bool bKeyDown, bool bAltDown, void\* pUserContext);

void CALLBACK OnGUIEvent(UINT nEvent, int nControlID, CDXUTControl\* pControl, void\* pUserContext);

bool CALLBACK IsD3D10DeviceAcceptable(UINT Adapter, UINT Output, D3D10\_DRIVER\_TYPE DeviceType, DXGI\_FORMAT BackBufferFormat, bool bWindowed, void\* pUserContext);

HRESULT CALLBACK OnD3D10CreateDevice(ID3D10Device\* pd3dDevice, const DXGI\_SURFACE\_DESC\* pBackBufferSurfaceDesc, void\* pUserContext);

HRESULT CALLBACK OnD3D10ResizedSwapChain(ID3D10Device\* pd3dDevice, IDXGISwapChain \*pSwapChain, const DXGI\_SURFACE\_DESC\* pBackBufferSurfaceDesc, void\* pUserContext);

void CALLBACK OnD3D10FrameRender(ID3D10Device\* pd3dDevice, double time\_d, float elapsedTime, void\* pUserContext);

void CALLBACK OnD3D10SwapChainReleasing(void\* pUserContext);

void CALLBACK OnD3D10DestroyDevice(void\* pUserContext);

void InitApp();

void SimulateCloth(ID3D10Device\*, float, float);

void ComputeNormalTangent(ID3D10Device\*);

void RenderScene(ID3D10Device\*, D3DXMATRIX&, D3DXMATRIX\* = 0);

void SetRenderState(D3DXMATRIX&, ID3D10ShaderResourceView\*, ID3D10ShaderResourceView\*, const D3DXMATRIX&, const D3DXMATRIX\*);

void SetMatrices(D3DXMATRIX&, const D3DXMATRIX&, const D3DXMATRIX\*);

void SetMaterial(bool);

void ComputeThinningMatrix(D3DXMATRIX&, float, D3DXMATRIX&);

void RenderTangentSpace(ID3D10Device\*);

//void RenderText();

void RenderPosition(ID3D10Device\*, ID3D10RenderTargetView\*, int, int, int, D3DXMATRIX\*);

D3DXMATRIX GetPixelTargetTransform(int, int);

void WorldToScreen(const D3DXVECTOR3&, float&, float&);

D3DXVECTOR3 ScreenToTrackballPoint(float, float);

void MoveCloth(HWND, int, WPARAM, LPARAM);

bool CutCloth(ID3D10Device\*, HWND = 0, int = 0, LPARAM = 0);

void RenderCutterEdge(ID3D10Device\*, int, int, int, int);

void InitializePositions(Particle\*);

HRESULT CreateIndexBuffers(ID3D10Device\*, Particle\*);

HRESULT CreateSimulationIB(ID3D10Device\*, Particle\*);

HRESULT CreateTangentSpaceIB(ID3D10Device\*);

HRESULT CreateRenderingIB(ID3D10Device\*);

HRESULT CreateVertexBuffers(ID3D10Device\*, Particle\*);

HRESULT CreatePositionVB(ID3D10Device\*, Particle\*);

HRESULT CreateTexCoordVB(ID3D10Device\*);

HRESULT CreateParticleVB(ID3D10Device\*);

HRESULT CreateTangentSpaceVB(ID3D10Device\*);

HRESULT CreateCutterEdgeVB(ID3D10Device\*);

HRESULT CreateAnchorVB(ID3D10Device\*);

HRESULT CreateInputLayouts(ID3D10Device\*);

HRESULT CreateApplyForcesIL(ID3D10Device\*);

HRESULT CreateSatisfyConstraintsIL(ID3D10Device\*);

HRESULT CreateRenderClothIL(ID3D10Device\*);

HRESULT CreateRenderTangentSpaceIL(ID3D10Device\*);

HRESULT CreateTextures(ID3D10Device\*);

HRESULT CreateRasterizerStates(ID3D10Device\*);

void InitPlanes();

HRESULT CreatePlaneMesh(ID3D10Device\*);

void InitSpheres();

HRESULT CreateSphereMesh(ID3D10Device\*);

HRESULT CreateShapeMesh(ID3D10Device\*, ID3DX10Mesh\*\*, MeshVertex\* vertices, UINT, WORD\*, UINT);

//--------------------------------------------------------------------------------------

// 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)

{

// Enable run-time memory check for debug builds.

#if defined(DEBUG) | defined(\_DEBUG)

\_CrtSetDbgFlag(\_CRTDBG\_ALLOC\_MEM\_DF | \_CRTDBG\_LEAK\_CHECK\_DF);

#endif

// DXUT will create and use the best device (either D3D9 or D3D10)

// that is available on the system depending on which D3D callbacks are set below

// Set DXUT callbacks

DXUTSetCallbackDeviceChanging(ModifyDeviceSettings);

DXUTSetCallbackMsgProc(MsgProc);

DXUTSetCallbackKeyboard(OnKeyboard);

DXUTSetCallbackFrameMove(OnFrameMove);

DXUTSetCallbackD3D10DeviceAcceptable(IsD3D10DeviceAcceptable);

DXUTSetCallbackD3D10DeviceCreated(OnD3D10CreateDevice);

DXUTSetCallbackD3D10SwapChainResized(OnD3D10ResizedSwapChain);

DXUTSetCallbackD3D10FrameRender(OnD3D10FrameRender);

DXUTSetCallbackD3D10SwapChainReleasing(OnD3D10SwapChainReleasing);

DXUTSetCallbackD3D10DeviceDestroyed(OnD3D10DestroyDevice);

InitApp();

DXUTInit(true, true, NULL); // Parse the command line, show msgboxes on error, no extra command line params

DXUTSetCursorSettings(true, true); // Show the cursor and clip it when in full screen

DXUTCreateWindow(L"Cloth");

DXUTCreateDevice(true, 800, 600);

DXUTMainLoop(); // Enter into the DXUT render loop

return DXUTGetExitCode();

}

//--------------------------------------------------------------------------------------

// Initialize the app

//--------------------------------------------------------------------------------------

void InitApp()

{

g\_SettingsDlg.Init(&g\_DialogResourceManager);

g\_HUD.Init(&g\_DialogResourceManager);

g\_SampleUI.Init(&g\_DialogResourceManager);

g\_HUD.SetCallback(OnGUIEvent);

int iY = 10;

int width = 150;

int height = 22;

WCHAR label[MAX\_PATH];

g\_SampleUI.SetCallback(OnGUIEvent);

iY = 5;

int sliderOffset = 15;

int sliderWidth = 130;

StringCchPrintf(label, MAX\_PATH, L"Simulation passes: %d", g\_NumSimPasses);

g\_SampleUI.AddStatic(IDC\_NUM\_SIM\_PASSES\_LABEL, label, 0, iY += 24, width, height);

g\_SampleUI.AddSlider(IDC\_NUM\_SIM\_PASSES, sliderOffset, iY += 24, sliderWidth, height, 1, 10, g\_NumSimPasses);

StringCchPrintf(label, MAX\_PATH, L"Wind heading: %d", g\_WindHeading);

g\_SampleUI.AddStatic(IDC\_WIND\_HEADING\_LABEL, label, 0, iY += 24, width, height);

g\_SampleUI.AddSlider(IDC\_WIND\_HEADING, sliderOffset, iY += 24, sliderWidth, height, 1, 360, g\_WindHeading);

StringCchPrintf(label, MAX\_PATH, L"Wind strength: %d", g\_WindStrength);

g\_SampleUI.AddStatic(IDC\_WIND\_STRENGTH\_LABEL, label, 0, iY += 24, width, height);

g\_SampleUI.AddSlider(IDC\_WIND\_STRENGTH, sliderOffset, iY += 24, sliderWidth, height, 0, 60, g\_WindStrength);

InitPlanes();

InitSpheres();

}

void SimulateCloth(ID3D10Device\* pd3dDevice, float timeStep, float oldTimeStep)

{

HRESULT hr = S\_OK;

// Effect variables

V(g\_Effect->GetVariableByName("TimeStep")->AsScalar()->SetFloat(timeStep));

V(g\_Effect->GetVariableByName("OldTimeStep")->AsScalar()->SetFloat(oldTimeStep));

D3DXVECTOR2 springConstraint(2.0f / (g\_Width - 1), 2.0f / (g\_Height - 1));

V(g\_Effect->GetVariableByName("DistanceAtRestX")->AsScalar()->SetFloat(springConstraint.x));

V(g\_Effect->GetVariableByName("DistanceAtRestY")->AsScalar()->SetFloat(springConstraint.y));

float springConstraintLength = D3DXVec2Length(&springConstraint);

V(g\_Effect->GetVariableByName("DistanceAtRestXY")->AsScalar()->SetFloat(springConstraintLength));

V(g\_Effect->GetVariableByName("LastPrimitiveIndex")->AsScalar()->SetInt(g\_LastPrimitiveIndex));

V(g\_Effect->GetVariableByName("LastPrimitiveSize")->AsScalar()->SetInt(g\_LastPrimitiveSize));

// Planes

V(g\_Effect->GetVariableByName("NumPlanes")->AsScalar()->SetInt(g\_NumPlanes));

V(g\_Effect->GetVariableByName("PlaneList")->SetRawValue(g\_Plane, 0, sizeof(g\_Plane)));

// Spheres

V(g\_Effect->GetVariableByName("NumSpheres")->AsScalar()->SetInt(g\_NumSpheres));

V(g\_Effect->GetVariableByName("SphereList")->SetRawValue(g\_Sphere, 0, sizeof(g\_Sphere)));

ID3D10EffectTechnique\* technique;

ID3D10Buffer\* buffers[] = { 0, 0, 0 };

UINT strides[] = { 0, 0, 0 };

UINT offset[] = { 0, 0, 0 };

// Forces

if (g\_Reset) {

pd3dDevice->CopyResource(g\_ParticleVB[g\_Old], g\_InitialPositionVB);

pd3dDevice->CopyResource(g\_ParticleVB[g\_Current], g\_InitialPositionVB);

ComputeNormalTangent(pd3dDevice);

InitializePositions(0);

g\_Reset = false;

}

buffers[0] = g\_ParticleVB[g\_Current];

buffers[1] = g\_ParticleVB[g\_Old];

buffers[2] = g\_NormalTangentVB;

strides[0] = strides[1] = sizeof(Particle);

strides[2] = sizeof(NormalTangent);

pd3dDevice->IASetVertexBuffers(0, 3, buffers, strides, offset);

pd3dDevice->IASetInputLayout(g\_ApplyForcesIL);

pd3dDevice->SOSetTargets(1, &g\_ParticleVB[g\_New], offset);

technique = g\_Effect->GetTechniqueByName("SimulateCloth");

pd3dDevice->IASetPrimitiveTopology(D3D10\_PRIMITIVE\_TOPOLOGY\_POINTLIST);

float heading = D3DXToRadian(g\_WindHeading);

D3DXVECTOR3 wind(sinf(heading), 0, cosf(heading));

wind \*= (float)g\_WindStrength;

V(g\_Effect->GetVariableByName("Wind")->AsVector()->SetFloatVector(wind));

technique->GetPassByName("ApplyForces")->Apply(0);

pd3dDevice->Draw(g\_NumVertices, 0);

pd3dDevice->SOSetTargets(0, 0, 0);

// Rotate buffers

int discard = g\_Old;

g\_Old = g\_Current;

g\_Current = g\_New;

g\_New = discard;

// Move anchor points

if (g\_TransformAnchorPoints) {

UINT strides = sizeof(AnchorPoint);

UINT offset = 0;

pd3dDevice->IASetVertexBuffers(0, 1, &g\_AnchorVB, &strides, &offset);

V(g\_Effect->GetVariableByName("AnchorPoints")->AsShaderResource()->SetResource(g\_ParticleSRV[g\_Old]));

V(g\_Effect->GetVariableByName("AnchorPointTransform")->AsMatrix()->SetMatrix(g\_Transform));

technique->GetPassByName("TransformAnchorPoints")->Apply(0);

pd3dDevice->IASetInputLayout(g\_AnchorIL);

pd3dDevice->IASetPrimitiveTopology(D3D10\_PRIMITIVE\_TOPOLOGY\_POINTLIST);

pd3dDevice->OMSetRenderTargets(1, &g\_ParticleRTV[g\_Current], 0);

D3D10\_VIEWPORT viewPort;

viewPort.TopLeftX = 0;

viewPort.TopLeftY = 0;

viewPort.Width = g\_NumVertices;

viewPort.Height = 1;

viewPort.MinDepth = 0;

viewPort.MaxDepth = 1;

pd3dDevice->RSSetViewports(1, &viewPort);

pd3dDevice->Draw(g\_NumAnchorPoints, 0);

V(g\_Effect->GetVariableByName("AnchorPoints")->AsShaderResource()->SetResource(0));

pd3dDevice->RSSetViewports(1, &g\_BackBufferVP);

pd3dDevice->OMSetRenderTargets(1, &g\_BackBufferRTV, g\_BackBufferDSV);

g\_TransformAnchorPoints = false;

D3DXVECTOR3 newClothCenter;

D3DXVec3TransformCoord(&newClothCenter, &g\_Center, &g\_Transform);

g\_Center = newClothCenter;

}

// Compute cutter

if (CutCloth(pd3dDevice)) {

V(g\_Effect->GetVariableByName("Eye")->SetRawValue(&g\_Eye, 0, sizeof(D3DXVECTOR3)));

pd3dDevice->CopyResource(g\_CutterEdgeCB, g\_CutterEdgeVB);

V(g\_Effect->GetVariableByName("Cut")->AsScalar()->SetBool(true));

V(g\_Effect->GetVariableByName("NumCutterTriangles")->AsScalar()->SetInt(g\_NumCutterTriangles));

V(g\_Effect->GetVariableByName("FirstCutterTriangle")->AsScalar()->SetInt(g\_FirstCutterTriangle));

}

// Uncut

if (g\_Uncut) {

buffers[0] = g\_ParticleVB[g\_Current];

buffers[1] = 0;

buffers[2] = 0;

strides[0] = sizeof(Particle);

pd3dDevice->IASetVertexBuffers(0, 3, buffers, strides, offset);

pd3dDevice->IASetInputLayout(g\_SatisfyConstraintsIL);

pd3dDevice->SOSetTargets(1, &g\_ParticleVB[g\_New], offset);

technique = g\_Effect->GetTechniqueByName("SimulateCloth");

assert(technique->IsValid());

pd3dDevice->IASetPrimitiveTopology(D3D10\_PRIMITIVE\_TOPOLOGY\_POINTLIST);

technique->GetPassByName("Uncut")->Apply(0);

pd3dDevice->Draw(g\_NumVertices, 0);

pd3dDevice->SOSetTargets(0, 0, 0);

g\_Uncut = false;

// Rotate buffers

discard = g\_Current;

g\_Current = g\_New;

g\_New = discard;

}

// Constraints

pd3dDevice->IASetInputLayout(g\_SatisfyConstraintsIL);

for (int simPass = 0; simPass < g\_NumSimPasses; ++simPass) {

for (int p = PASS\_SIMULATE\_STRUCTURAL\_AND\_SHEAR\_SPRING\_0; p <= PASS\_SIMULATE\_COLLISION; ++p) {

// Swap VBs

pd3dDevice->SOSetTargets(0, 0, 0);

buffers[0] = g\_ParticleVB[g\_Current];

buffers[1] = 0;

buffers[2] = 0;

pd3dDevice->IASetVertexBuffers(0, 3, buffers, strides, offset);

pd3dDevice->SOSetTargets(1, &g\_ParticleVB[g\_New], offset);

int indexCount;

if (p == PASS\_SIMULATE\_COLLISION) {

pd3dDevice->IASetPrimitiveTopology(D3D10\_PRIMITIVE\_TOPOLOGY\_POINTLIST);

indexCount = g\_NumVertices;

}

else {

pd3dDevice->IASetPrimitiveTopology(D3D10\_PRIMITIVE\_TOPOLOGY\_LINELIST\_ADJ);

indexCount = g\_NumSimIndices;

}

// Draw

pd3dDevice->IASetIndexBuffer(g\_SimulateIB[p], DXGI\_FORMAT\_R16\_UINT, 0);

V(g\_Effect->GetVariableByName("NumSimulatedPrimitives")->AsScalar()->SetInt(g\_NumSimulatedPrimitives[p]));

ID3D10EffectPass\* pPass = NULL;

if (p <= PASS\_SIMULATE\_STRUCTURAL\_AND\_SHEAR\_SPRING\_1) {

V(g\_Effect->GetVariableByName("ConnectionConfigOffset")->AsScalar()->SetInt(p == PASS\_SIMULATE\_STRUCTURAL\_AND\_SHEAR\_SPRING\_0 ? 0 : 8));

pPass = technique->GetPassByName("SatisfyStructuralAndShearSpringConstraints");

}

else if (p <= PASS\_SIMULATE\_SHEAR\_SPRING\_1)

{

pPass = technique->GetPassByName("SatisfyShearSpringConstraints");

}

else

{

pPass = technique->GetPassByName("SatisfyCollisionConstraints");

}

assert(pPass != NULL);

assert(pPass->IsValid());

pPass->Apply(0);

pd3dDevice->DrawIndexed(indexCount, 0, 0);

// Rotate buffers

discard = g\_Current;

g\_Current = g\_New;

g\_New = discard;

}

V(g\_Effect->GetVariableByName("Cut")->AsScalar()->SetBool(false));

}

pd3dDevice->SOSetTargets(0, 0, 0);

// Normals and tangents

ComputeNormalTangent(pd3dDevice);

}

void ComputeNormalTangent(ID3D10Device\* pd3dDevice)

{

ID3D10EffectTechnique\* technique;

ID3D10Buffer\* buffers[] = { 0, 0, 0 };

UINT strides[] = { 0, 0, 0 };

UINT offset[] = { 0, 0, 0 };

strides[0] = sizeof(Particle);

buffers[0] = g\_ParticleVB[g\_Current];

buffers[1] = 0;

buffers[2] = 0;

pd3dDevice->IASetVertexBuffers(0, 3, buffers, strides, offset);

pd3dDevice->IASetIndexBuffer(g\_NormalTangentIB, DXGI\_FORMAT\_R16\_UINT, 0);

pd3dDevice->IASetInputLayout(g\_SatisfyConstraintsIL);

pd3dDevice->SOSetTargets(1, &g\_NormalTangentVB, offset);

technique = g\_Effect->GetTechniqueByName("SimulateCloth");

assert(technique->IsValid());

pd3dDevice->IASetPrimitiveTopology(D3D10\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST\_ADJ);

technique->GetPassByName("ComputeNormalTangent")->Apply(0);

pd3dDevice->DrawIndexed(6 \* g\_NumVertices, 0, 0);

pd3dDevice->SOSetTargets(0, 0, 0);

}

//--------------------------------------------------------------------------------------

// Render

//--------------------------------------------------------------------------------------

void CALLBACK OnD3D10FrameRender(ID3D10Device\* pd3dDevice, double time\_d, float elapsedTime, void\* pUserContext)

{

// Clear the depth stencil

float clearColor[] = { 0.176f, 0.196f, 0.667f, 0 };

pd3dDevice->ClearRenderTargetView(DXUTGetD3D10RenderTargetView(), clearColor);

pd3dDevice->ClearDepthStencilView(DXUTGetD3D10DepthStencilView(), D3D10\_CLEAR\_DEPTH, 1.0, 0);

// If the settings dialog is being shown, then render it instead of rendering the app's scene

if(g\_SettingsDlg.IsActive())

{

g\_SettingsDlg.OnRender(elapsedTime);

return;

}

// Get the projection & view matrix from the camera class

g\_Projection = \*g\_Camera.GetProjMatrix();

g\_View = \*g\_Camera.GetViewMatrix();

D3DXMATRIX viewInv;

D3DXMatrixInverse(&viewInv, 0, &g\_View);

g\_Eye = D3DXVECTOR3(viewInv(3, 0), viewInv(3, 1), viewInv(3, 2));

// View projection matrix

g\_ViewProjection = g\_View \* g\_Projection;

// Render scene

RenderScene(pd3dDevice, g\_ViewProjection);

// Render tangent space

if (g\_ShowTangentSpace)

RenderTangentSpace(pd3dDevice);

// HUD

DXUT\_BeginPerfEvent( DXUT\_PERFEVENTCOLOR, L"HUD / Stats" );

g\_SampleUI.OnRender( elapsedTime );

g\_HUD.OnRender( elapsedTime );

DXUT\_EndPerfEvent();

//RenderText();

}

void RenderScene(ID3D10Device\* pd3dDevice, D3DXMATRIX& viewProjection, D3DXMATRIX\* renderPositionTransform)

{

// Render scene

pd3dDevice->IASetInputLayout(g\_RenderMeshIL);

for (int i = 0; i < g\_NumPlanes; ++i) {

SetRenderState(g\_PlaneWorld[i], g\_PlaneDiffuseTexture[i], g\_PlaneNormalMap[i], viewProjection, renderPositionTransform);

g\_PlaneMesh->DrawSubset(0);

}

for (int i = 0; i < g\_NumSpheres; ++i) {

SetRenderState(g\_SphereWorld[i], 0, 0, viewProjection, renderPositionTransform);

g\_SphereMesh->DrawSubset(0);

}

HRESULT hr = S\_OK;

// Render cloth

ID3D10Buffer\* buffers[] = { 0, 0, 0 };

UINT strides[] = { 0, 0, 0 };

UINT offset[] = { 0, 0, 0 };

V(g\_Effect->GetVariableByName("ViewProjection")->AsMatrix()->SetMatrix(viewProjection));

if (renderPositionTransform)

V(g\_Effect->GetVariableByName("Transform")->AsMatrix()->SetMatrix(\*renderPositionTransform));

V(g\_Effect->GetVariableByName("DiffuseCoeff")->AsScalar()->SetFloat(1.0f));

V(g\_Effect->GetVariableByName("SpecularCoeff")->AsScalar()->SetFloat(0.1f));

V(g\_Effect->GetVariableByName("SpecularPower")->AsScalar()->SetFloat(10));

V(g\_Effect->GetVariableByName("Color")->AsVector()->SetFloatVector(D3DXVECTOR3(1, 1, 1)));

buffers[0] = g\_ParticleVB[g\_Current];

buffers[1] = g\_NormalTangentVB;

buffers[2] = g\_TexCoordVB;

strides[0] = sizeof(Particle);

strides[1] = sizeof(NormalTangent);

strides[2] = sizeof(TexCoord);

pd3dDevice->IASetVertexBuffers(0, 3, buffers, strides, offset);

pd3dDevice->IASetInputLayout(g\_RenderClothIL);

pd3dDevice->IASetIndexBuffer(g\_RenderIB, DXGI\_FORMAT\_R16\_UINT, 0);

pd3dDevice->IASetPrimitiveTopology(D3D10\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST);

ID3D10EffectTechnique\* technique = g\_Effect->GetTechniqueByName("RenderCloth");

assert(technique->IsValid());

V(g\_Effect->GetVariableByName("PositionStepX")->AsScalar()->SetFloat(1.0f / (g\_Width - 1)));

V(g\_Effect->GetVariableByName("PositionStepY")->AsScalar()->SetFloat(1.0f / (g\_Height - 1)));

V(g\_Effect->GetVariableByName("TexCoordStepX")->AsScalar()->SetFloat(0.5f \* g\_TexCoordScale / (g\_Width - 1)));

V(g\_Effect->GetVariableByName("TexCoordStepY")->AsScalar()->SetFloat(0.5f \* g\_TexCoordScale / (g\_Height - 1)));

if (renderPositionTransform)

technique->GetPassByName("Position")->Apply(0);

else if (g\_Wireframe)

technique->GetPassByName("Colored")->Apply(0);

else {

V(g\_Effect->GetVariableByName("DiffuseTexture")->AsShaderResource()->SetResource(g\_ClothDiffuseTexture));

V(g\_Effect->GetVariableByName("NormalMap")->AsShaderResource()->SetResource(g\_ClothNormalMap));

technique->GetPassByName("Textured")->Apply(0);

}

bool renderInWireframe = (renderPositionTransform == 0 && g\_Wireframe);

if (renderInWireframe)

pd3dDevice->RSSetState(g\_WireframeRS);

pd3dDevice->DrawIndexed(g\_NumIndices, 0, 0);

if (renderInWireframe)

pd3dDevice->RSSetState(g\_SolidRS);

}

void SetRenderState(D3DXMATRIX& world, ID3D10ShaderResourceView\* diffuseTexture, ID3D10ShaderResourceView\* normalMap,

const D3DXMATRIX& viewProjection, const D3DXMATRIX\* renderPositionTransform)

{

HRESULT hr = S\_OK;

SetMatrices(world, viewProjection, renderPositionTransform);

V(g\_Effect->GetVariableByName("Eye")->SetRawValue(&g\_Eye, 0, sizeof(D3DXVECTOR3)));

SetMaterial(diffuseTexture == 0);

ID3D10EffectTechnique\* technique = g\_Effect->GetTechniqueByName("RenderMesh");

assert(technique->IsValid());

if (renderPositionTransform)

technique->GetPassByName("Position")->Apply(0);

else if (diffuseTexture) {

V(g\_Effect->GetVariableByName("DiffuseTexture")->AsShaderResource()->SetResource(diffuseTexture));

V(g\_Effect->GetVariableByName("NormalMap")->AsShaderResource()->SetResource(normalMap));

technique->GetPassByName("Textured")->Apply(0);

}

else

technique->GetPassByName("Untextured")->Apply(0);

}

void SetMatrices(D3DXMATRIX& world, const D3DXMATRIX& viewProjection, const D3DXMATRIX\* renderPositionTransform)

{

HRESULT hr = S\_OK;

V(g\_Effect->GetVariableByName("World")->AsMatrix()->SetMatrix(world));

D3DXMATRIX worldI;

D3DXMatrixInverse(&worldI, 0, &world);

D3DXMATRIX worldIT;

D3DXMatrixTranspose(&worldIT, &worldI);

V(g\_Effect->GetVariableByName("WorldIT")->AsMatrix()->SetMatrix(worldIT));

D3DXMATRIX thinningMatrix;

ComputeThinningMatrix(world, -0.065f, thinningMatrix);

D3DXMATRIX worldViewProjection = thinningMatrix \* world \* viewProjection;

V(g\_Effect->GetVariableByName("WorldViewProjection")->AsMatrix()->SetMatrix(worldViewProjection));

if (renderPositionTransform) {

D3DXMATRIX transform = world \* \*renderPositionTransform;

V(g\_Effect->GetVariableByName("Transform")->AsMatrix()->SetMatrix(transform));

}

}

void SetMaterial(bool shiny)

{

HRESULT hr = S\_OK;

if (shiny) {

V(g\_Effect->GetVariableByName("DiffuseCoeff")->AsScalar()->SetFloat(1.0f));

V(g\_Effect->GetVariableByName("SpecularCoeff")->AsScalar()->SetFloat(0.4f));

V(g\_Effect->GetVariableByName("SpecularPower")->AsScalar()->SetFloat(1000));

}

else {

V(g\_Effect->GetVariableByName("DiffuseCoeff")->AsScalar()->SetFloat(0.6f));

V(g\_Effect->GetVariableByName("SpecularCoeff")->AsScalar()->SetFloat(0.2f));

V(g\_Effect->GetVariableByName("SpecularPower")->AsScalar()->SetFloat(10.0f));

}

V(g\_Effect->GetVariableByName("Color")->AsVector()->SetFloatVector(D3DXVECTOR3(1, 1, 1)));

}

void ComputeThinningMatrix(D3DXMATRIX& world, float thinning, D3DXMATRIX& mat)

{

D3DXVECTOR3 dim;

for (int i = 0; i < 3; ++i) {

D3DXVECTOR3 row(world(i, 0), world(i, 1), world(i, 2));

dim[i] = D3DXVec3Length(&row);

}

D3DXMatrixScaling(&mat, max(1 + thinning / dim[0], 0), max(1 + thinning / dim[1], 0), max(1 + thinning / dim[2], 0));

}

void RenderTangentSpace(ID3D10Device\* pd3dDevice)

{

ID3D10EffectTechnique\* technique;

ID3D10Buffer\* buffers[] = { 0, 0, 0 };

UINT strides[] = { 0, 0, 0 };

UINT offset[] = { 0, 0, 0 };

HRESULT hr = S\_OK;

buffers[0] = g\_ParticleVB[g\_Current];

buffers[1] = g\_NormalTangentVB;

buffers[2] = 0;

strides[0] = sizeof(Particle);

strides[1] = sizeof(NormalTangent);

pd3dDevice->IASetVertexBuffers(0, 2, buffers, strides, offset);

technique = g\_Effect->GetTechniqueByName("RenderTangentSpace");

assert(technique->IsValid());

V(g\_Effect->GetVariableByName("ViewProjection")->AsMatrix()->SetMatrix(g\_ViewProjection));

pd3dDevice->IASetInputLayout(g\_RenderTangentSpaceIL);

pd3dDevice->IASetPrimitiveTopology(D3D10\_PRIMITIVE\_TOPOLOGY\_POINTLIST);

technique->GetPassByIndex(0)->Apply(0);

pd3dDevice->Draw(g\_NumVertices, 0);

}

void MoveCloth(HWND hWnd, int msg, WPARAM wParam, LPARAM lParam)

{

static LPARAM g\_LastMouse;

static D3DXVECTOR3 g\_LastTrackballPoint;

static bool g\_ClothIsMoved;

static D3DXMATRIX g\_PivotMatrix, g\_PivotMatrixInv;

static float g\_ScreenClothCenterX, g\_ScreenClothCenterY;

int mouseX = (short)LOWORD(lParam);

int mouseY = (short)HIWORD(lParam);

switch (msg) {

case WM\_LBUTTONDOWN:

case WM\_MBUTTONDOWN:

// Pivot transform

{

D3DXMATRIX pivotTranslation;

D3DXMatrixTranslation(&pivotTranslation, -g\_Center.x, -g\_Center.y, -g\_Center.z);

D3DXMATRIX pivotRotation = g\_View;

pivotRotation(3, 0) = pivotRotation(3, 1) = pivotRotation(3, 2) = 0;

g\_PivotMatrix = pivotTranslation \* pivotRotation;

D3DXMatrixInverse(&g\_PivotMatrixInv, 0, &g\_PivotMatrix);

}

WorldToScreen(g\_Center, g\_ScreenClothCenterX, g\_ScreenClothCenterY);

g\_LastTrackballPoint = ScreenToTrackballPoint(mouseX - g\_ScreenClothCenterX, mouseY - g\_ScreenClothCenterY);

g\_LastMouse = lParam;

g\_ClothIsMoved = true;

break;

case WM\_MOUSEMOVE:

if (g\_ClothIsMoved) {

D3DXMATRIX transform;

D3DXMatrixIdentity(&transform);

// Translate

if (wParam & MK\_LBUTTON) {

float scale = 2;

float dx = (scale \* ((short)LOWORD(g\_LastMouse) - mouseX)) / g\_BackBufferWidth;

float dy = (scale \* ((short)HIWORD(g\_LastMouse) - mouseY)) / g\_BackBufferHeight;

if (g\_ShiftDown)

D3DXMatrixTranslation(&transform, 0, 0, dy);

else

D3DXMatrixTranslation(&transform, -dx, dy, 0);

g\_LastMouse = lParam;

}

// Rotate

else if (wParam & MK\_MBUTTON) {

D3DXVECTOR3 trackballPoint = ScreenToTrackballPoint(mouseX - g\_ScreenClothCenterX, mouseY - g\_ScreenClothCenterY);

D3DXVECTOR3 cross;

D3DXVec3Cross(&cross, &g\_LastTrackballPoint, &trackballPoint);

float dot = D3DXVec3Dot(&g\_LastTrackballPoint, &trackballPoint);

D3DXQUATERNION dq(cross.x, cross.y, cross.z, dot);

D3DXMatrixRotationQuaternion(&transform, &dq);

g\_LastTrackballPoint = trackballPoint;

}

// Final transform

g\_Transform = g\_PivotMatrix \* transform \* g\_PivotMatrixInv;

g\_TransformAnchorPoints = true;

}

break;

case WM\_LBUTTONUP:

case WM\_MBUTTONUP:

g\_ClothIsMoved = false;

break;

default:

break;

}

}

bool CutCloth(ID3D10Device\* pd3dDevice, HWND hWnd, int msg, LPARAM lParam)

{

static bool g\_IsCut;

static LPARAM g\_LastMouse;

static LPARAM g\_CurrentMouse;

if (pd3dDevice) {

if (g\_LastMouse != g\_CurrentMouse) {

RenderCutterEdge(pd3dDevice,

LOWORD(g\_LastMouse), HIWORD(g\_LastMouse),

LOWORD(g\_CurrentMouse), HIWORD(g\_CurrentMouse));

g\_LastMouse = g\_CurrentMouse;

return true;

}

}

else

switch (msg) {

case WM\_LBUTTONDOWN:

g\_IsCut = true;

g\_LastMouse = lParam;

g\_CurrentMouse = lParam;

break;

case WM\_MOUSEMOVE:

if (g\_IsCut)

g\_CurrentMouse = lParam;

break;

case WM\_LBUTTONUP:

g\_IsCut = false;

g\_NumCutterTriangles = 0;

break;

default:

break;

}

return false;

}

//--------------------------------------------------------------------------------------

// Handle key presses

//--------------------------------------------------------------------------------------

void CALLBACK OnKeyboard(UINT nChar, bool bKeyDown, bool bAltDown, void\* pUserContext)

{

if (bKeyDown)

switch(nChar)

{

case '0':

g\_Reset = true;

Step(1);

break;

case 'S':

Step(1);

break;

case VK\_F1:

g\_Help = !g\_Help;

break;

case VK\_SPACE:

g\_Run = !g\_Run;

break;

case 'Z':

g\_Uncut = true;

break;

case 'W':

g\_Wireframe = !g\_Wireframe;

break;

case VK\_SHIFT:

g\_ShiftDown = true;

break;

case VK\_CONTROL:

g\_CtrlDown = true;

break;

}

else

switch(nChar)

{

case VK\_SHIFT:

g\_ShiftDown = false;

break;

case VK\_CONTROL:

g\_CtrlDown = false;

break;

}

}

//--------------------------------------------------------------------------------------

// Initialize positions

//--------------------------------------------------------------------------------------

void InitializePositions(Particle\* particles)

{

g\_Center = D3DXVECTOR3(0, 0, 0);

int v = 0;

for (int z = 0; z < g\_Height; ++z)

for (int x = 0; x < g\_Width; ++x) {

D3DXVECTOR3 position;

position.x = 2 \* x / static\_cast<float>(g\_Width - 1) - 1;

position.y = 1;

position.z = 2 \* z / static\_cast<float>(g\_Height - 1) - 1;

if (particles) {

particles[v].Position = position;

particles[v].State = 0;

SetFree(particles[v]);

ResetConnectivity(particles[v]);

}

for (int i = 0; i < g\_NumAnchorPoints; ++i)

if (g\_AnchorPoint[i].x == x && g\_AnchorPoint[i].y == z) {

if (particles)

UnsetFree(particles[v]);

g\_Center += position;

}

++v;

}

g\_Center /= (float)g\_NumAnchorPoints;

D3DXMatrixIdentity(&g\_Transform);

g\_TransformAnchorPoints = false;

}

//--------------------------------------------------------------------------------------

//--------------------------------------------------------------------------------------

// Create textures

//--------------------------------------------------------------------------------------

HRESULT CreateTextures(ID3D10Device\* pd3dDevice)

{

HRESULT hr;

WCHAR str[MAX\_PATH];

// Cloth

V\_RETURN(NVUTFindDXSDKMediaFileCch(str, MAX\_PATH, L"..\\..\\Media\\Texture.dds"));

V\_RETURN(D3DX10CreateShaderResourceViewFromFile(pd3dDevice, str, 0, 0, &g\_ClothDiffuseTexture, &hr));

V\_RETURN(NVUTFindDXSDKMediaFileCch(str, MAX\_PATH, L"..\\..\\Media\\Texture\_normal.dds"));

V\_RETURN(D3DX10CreateShaderResourceViewFromFile(pd3dDevice, str, 0, 0, &g\_ClothNormalMap, &hr));

// Floor

V\_RETURN(NVUTFindDXSDKMediaFileCch(str, MAX\_PATH, L"..\\..\\Media\\lichen6.dds"));

V\_RETURN(D3DX10CreateShaderResourceViewFromFile(pd3dDevice, str, 0, 0, &g\_CellDiffuseTexture[0], &hr));

V\_RETURN(NVUTFindDXSDKMediaFileCch(str, MAX\_PATH, L"..\\..\\Media\\lichen6\_normal.dds"));

V\_RETURN(D3DX10CreateShaderResourceViewFromFile(pd3dDevice, str, 0, 0, &g\_CellNormalMap[0], &hr));

// Walls and ceiling

V\_RETURN(NVUTFindDXSDKMediaFileCch(str, MAX\_PATH, L"..\\..\\Media\\lichen1.dds"));

V\_RETURN(D3DX10CreateShaderResourceViewFromFile(pd3dDevice, str, 0, 0, &g\_CellDiffuseTexture[1], &hr));

V\_RETURN(NVUTFindDXSDKMediaFileCch(str, MAX\_PATH, L"..\\..\\Media\\lichen1\_normal.dds"));

V\_RETURN(D3DX10CreateShaderResourceViewFromFile(pd3dDevice, str, 0, 0, &g\_CellNormalMap[1], &hr));

g\_PlaneDiffuseTexture[0] = g\_CellDiffuseTexture[0];

g\_PlaneNormalMap[0] = g\_CellNormalMap[0];

for (int i = 1; i < g\_NumPlanes; ++i) {

g\_PlaneDiffuseTexture[i] = g\_CellDiffuseTexture[1];

g\_PlaneNormalMap[i] = g\_CellNormalMap[1];

}

return S\_OK;

}

//--------------------------------------------------------------------------------------

// Create scene

//--------------------------------------------------------------------------------------

void InitPlanes()

{

for (int i = 0; i < g\_NumPlanes; ++i) {

D3DXVECTOR3& normal = g\_Plane[i].Normal;

D3DXVECTOR3 p = -g\_Plane[i].Distance \* normal;

D3DXQUATERNION quat;

if (normal.x || normal.y || normal.z < 0) {

if (normal.x || normal.y) {

D3DXVECTOR3 v(-normal.y, normal.x, 0);

D3DXQuaternionRotationAxis(&quat, &v, acosf(normal.z));

}

else

quat = D3DXQUATERNION(1, 0, 0, 0);

D3DXMatrixAffineTransformation(&g\_PlaneWorld[i], 1, 0, &quat, &p);

}

else

D3DXMatrixTranslation(&g\_PlaneWorld[i], p.x, p.y, p.z);

float size = 7.5f;

D3DXMATRIX scaling;

D3DXMatrixScaling(&scaling, size, size, size);

g\_PlaneWorld[i] = scaling \* g\_PlaneWorld[i];

}

}

HRESULT CreatePlaneMesh(ID3D10Device\* pd3dDevice)

{

HRESULT hr;

UINT numIndices = 2 \* 3;

WORD\* indices = new WORD[numIndices];

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

indices[3] = 0; indices[4] = 3; indices[5] = 2;

UINT numVertices = 4;

MeshVertex\* vertices = new MeshVertex[numVertices];

float tile = 4;

vertices[0].Position = D3DXVECTOR3(-1, 1, 0);

vertices[0].Normal = D3DXVECTOR3(0, 0, 1);

vertices[0].TangentX = D3DXVECTOR3(1, 0, 0);

vertices[0].TexCoord = D3DXVECTOR2(0, 0);

vertices[1].Position = D3DXVECTOR3(1, 1, 0);

vertices[1].Normal = vertices[0].Normal;

vertices[1].TangentX = vertices[0].TangentX;

vertices[1].TexCoord = D3DXVECTOR2(tile, 0);

vertices[2].Position = D3DXVECTOR3(1, -1, 0);

vertices[2].Normal = vertices[0].Normal;

vertices[2].TangentX = vertices[0].TangentX;

vertices[2].TexCoord = D3DXVECTOR2(tile, tile);

vertices[3].Position = D3DXVECTOR3(-1, -1, 0);

vertices[3].Normal = vertices[0].Normal;

vertices[3].TangentX = vertices[0].TangentX;

vertices[3].TexCoord = D3DXVECTOR2(0, tile);

V\_RETURN(CreateShapeMesh(pd3dDevice, &g\_PlaneMesh, vertices, numVertices, indices, numIndices));

delete [] vertices;

delete [] indices;

return S\_OK;

}

void InitSpheres()

{

for (int i = 0; i < g\_NumSpheres; ++i) {

D3DXMatrixIdentity(&g\_SphereWorld[i]);

g\_SphereWorld[i](3, 0) = g\_Sphere[i].Center.x;

g\_SphereWorld[i](3, 1) = g\_Sphere[i].Center.y;

g\_SphereWorld[i](3, 2) = g\_Sphere[i].Center.z;

g\_SphereWorld[i](0, 0) = g\_Sphere[i].Radius;

g\_SphereWorld[i](1, 1) = g\_Sphere[i].Radius;

g\_SphereWorld[i](2, 2) = g\_Sphere[i].Radius;

}

}

HRESULT CreateSphereMesh(ID3D10Device\* pd3dDevice)

{

HRESULT hr;

if (g\_SphereMesh == 0) {

float radius[] = { 1, 1 };

V\_RETURN(CreateCapsuleMesh(pd3dDevice, 0, radius, 32, 32, &g\_SphereMesh));

}

return S\_OK;

}

//--------------------------------------------------------------------------------------

// Release D3D10 resources created in OnD3D10ResizedSwapChain

//--------------------------------------------------------------------------------------

void CALLBACK OnD3D10SwapChainReleasing(void\* pUserContext)

{

g\_DialogResourceManager.OnD3D10ReleasingSwapChain();

SAFE\_RELEASE(g\_BackBufferRTV);

SAFE\_RELEASE(g\_BackBufferDSV);

}

//--------------------------------------------------------------------------------------

// Release D3D10 resources created in OnD3D10CreateDevice

//--------------------------------------------------------------------------------------

void CALLBACK OnD3D10DestroyDevice(void\* pUserContext)

{

g\_DialogResourceManager.OnD3D10DestroyDevice();

g\_SettingsDlg.OnD3D10DestroyDevice();

SAFE\_RELEASE(g\_AnchorIL);

SAFE\_RELEASE(g\_ApplyForcesIL);

SAFE\_RELEASE(g\_SatisfyConstraintsIL);

SAFE\_RELEASE(g\_RenderTangentSpaceIL);

SAFE\_RELEASE(g\_RenderClothIL);

SAFE\_RELEASE(g\_InitialPositionVB);

SAFE\_RELEASE(g\_TexCoordVB);

SAFE\_RELEASE(g\_CutterEdgeVB);

SAFE\_RELEASE(g\_CutterEdgeCB);

SAFE\_RELEASE(g\_NormalTangentVB);

SAFE\_RELEASE(g\_AnchorVB);

for (int i = 0; i < 3; ++i) {

SAFE\_RELEASE(g\_ParticleVB[i]);

SAFE\_RELEASE(g\_ParticleSRV[i]);

SAFE\_RELEASE(g\_ParticleRTV[i]);

}

for (int i = 0; i < sizeof(g\_CellDiffuseTexture) / sizeof(g\_CellDiffuseTexture[0]); ++i)

SAFE\_RELEASE(g\_CellDiffuseTexture[i]);

for (int i = 0; i < sizeof(g\_CellNormalMap) / sizeof(g\_CellNormalMap[0]); ++i)

SAFE\_RELEASE(g\_CellNormalMap[i]);

SAFE\_RELEASE(g\_ClothDiffuseTexture);

SAFE\_RELEASE(g\_ClothNormalMap);

SAFE\_RELEASE(g\_CutterEdgeRTV);

for (int i = 0; i < NUM\_SIMULATE\_IB; ++i)

SAFE\_RELEASE(g\_SimulateIB[i]);

SAFE\_RELEASE(g\_NormalTangentIB);

SAFE\_RELEASE(g\_RenderIB);

SAFE\_RELEASE(g\_WireframeRS);

SAFE\_RELEASE(g\_SolidRS);

SAFE\_RELEASE(g\_Font);

SAFE\_RELEASE(g\_Sprite);

SAFE\_RELEASE(g\_Effect);

SAFE\_DELETE(g\_TxtHelper);

SAFE\_RELEASE(g\_RenderMeshIL);

SAFE\_RELEASE(g\_PlaneMesh);

SAFE\_RELEASE(g\_SphereMesh);

for (int i = 0; i < g\_NumCapsules; ++i)

SAFE\_RELEASE(g\_CapsuleMesh[i]);

}

