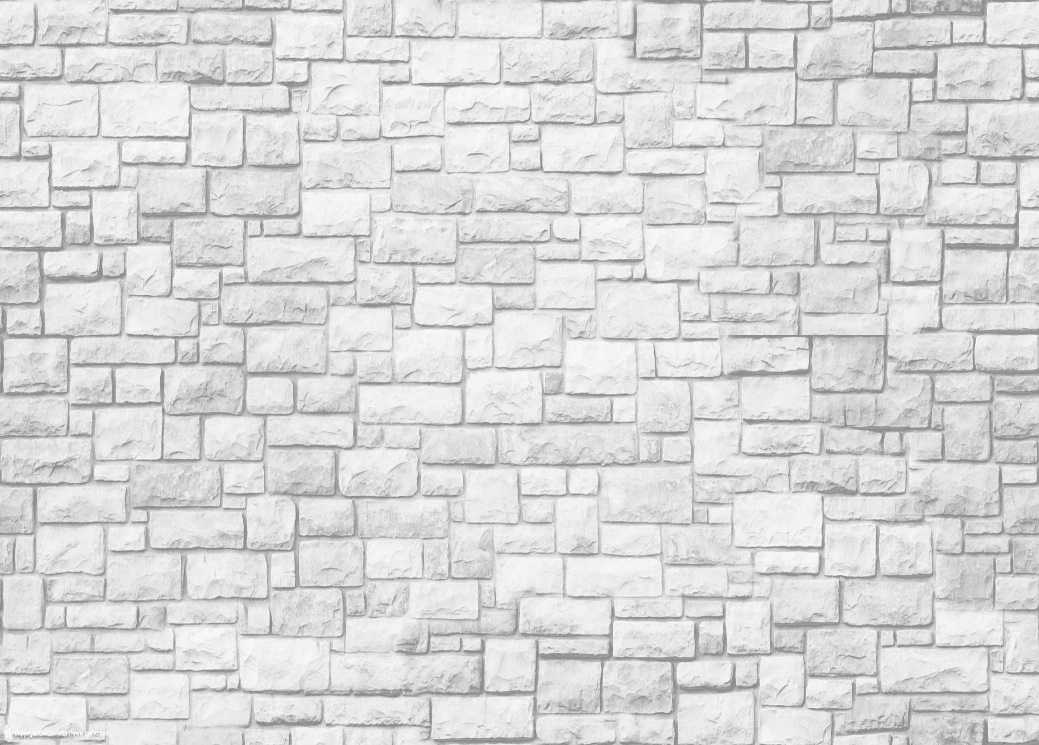
**Задание**

Построить модель плоской стены (плоский прямоугольник). Замостить модель текстурой стены, добавить текстуру для имитации шероховатостей и выщерблин. Подключить освещение различного типа на сцене: направленный источник света, «прожектор», точечный источник (в случае последних двух типов источников света ввести затухание).

**Выполнение лабораторной работы**

Создадим прямоугольную плоскость на координатах (-50,-40), (-50,40),(50,40),(50,-40) в плоскости z=0. Для этой плоскости используем текстуру стены из белого кирпича (Рис. 1).



**Рисунок 1.** Текстура стены из белого кирпича.

Затем перейдем к облицовке этого прямоугольника. Для этого создадим массив текстур для имитации шероховатостей и выщерблен.

private Texture[] cracks;

В процессе инициализации заполним этот массив изображениями в формате *.png* с текстурами поврежденной стены (Рис 2).

cracks = new Texture[4];

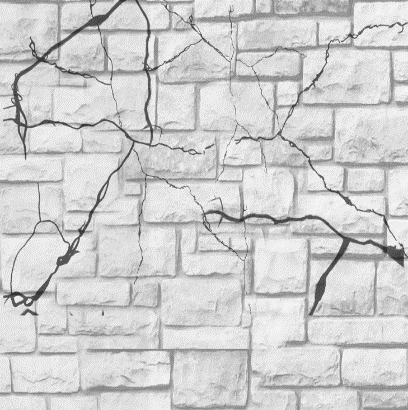
for (int i = 0; i <= 3; i++)

{

b = (Bitmap)System.Drawing.Image.FromFile("crack" + i + ".png");

cracks[i] = new Texture(device, b, 0, Pool.Managed);

}

**Рисунок 2.** Примеры текстур стены.

Задаем координаты вершин для соответствующих текстур частей стены. Все текстуры являются прямоугольниками, поэтому для всех них будет использоваться один и тот же индексный буфер.

ib\_1 = new IndexBuffer(typeof(int), 6, device, Usage.WriteOnly, Pool.Default);

indices = new int[]

{

2, 1, 0,

3, 2, 0

};

Поскольку мы работаем с текстурированными фигурами, нам нужно установить материал.

Затем мы будем циклически обновлять векторные и текстурные буферы для отображения примитивов каждой части стены.

device.SetStreamSource(0, cracks0\_vb, 0);

device.SetTexture(0, cracks[0]);

device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, 4, 0, 2);

…

device.SetStreamSource(0, cracks2\_vb, 0);

device.SetTexture(0, cracks[2]);

device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, 4, 0, 2);



**Рисунок 3.** Финальный результат при самоизлучении материала M.Emissive = Color.White.

Далее мы настроим освещение для нашей стены. Каждый источник света будет излучать разные цвета, чтобы их было легче различить.

Для направленного источника света мы сделаем его красным и зададим его направление как (0,0,-1).

Для точечного источника света мы разместим его в правом нижнем углу изображения, установим его дальность действия на 1f и цвет на синий. Затухание будет обратно пропорциональным квадратичным от расстояния.

Для прожекторного источника света мы разместим его в левом верхнем углу изображения, с направлением, параллельным направлению направленного источника света. Установим его дальность действия на 10f, цвет на зеленый и зададим гиперболическую зависимость от расстояния для затухания. Радиус внутреннего и внешнего конусов освещения будет равен 30f.



**Рисунок 4.** Финальный результат при выше заданных источниках освещения.

Как можно видеть на Рисунке 4, все источники света работают корректно, освещая стену.

При необходимости можно добавить вращение рисунка.

|  |
| --- |
| device.Transform.View = Matrix.LookAtLH(new Vector3((float)(100 \* Math.Sin(angle)), 0f, (float)(100 \* Math.Cos(angle))),  new Vector3(0, 0, 0),  new Vector3(0, 1, 0));  angle += 0.05f; |

**Приложение**

Код работы

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using Microsoft.DirectX.Direct3D;

using Microsoft.DirectX;

using Microsoft.VisualC;

namespace Directx\_Light

{

public partial class Form1 : Form

{

private Device device = null;

private VertexBuffer brickwall\_vb = null;

private VertexBuffer cracks0\_vb = null;

private VertexBuffer cracks1\_vb = null;

private VertexBuffer cracks2\_vb = null;

private VertexBuffer cracks3\_vb = null;

private CustomVertex.PositionNormalTextured[] brickwall\_vertices;

private CustomVertex.PositionNormalTextured[] cracks0\_vertices;

private CustomVertex.PositionNormalTextured[] cracks1\_vertices;

private CustomVertex.PositionNormalTextured[] cracks2\_vertices;

private CustomVertex.PositionNormalTextured[] cracks3\_vertices;

private IndexBuffer ib\_1 = null;

private int[] indices;

private float angle = 0;

private Bitmap b;

private Texture brickwall;

private Texture[] cracks;

public Form1()

{

InitializeComponent();

this.SetStyle(ControlStyles.AllPaintingInWmPaint | ControlStyles.Opaque, true);

InitializeDevice();

VertexDeclaration();

CameraPositioning();

}

public void InitializeDevice()

{

PresentParameters presentParams = new PresentParameters();

presentParams.Windowed = true;

presentParams.SwapEffect = SwapEffect.Discard;

device = new Device(0, DeviceType.Hardware, this, CreateFlags.SoftwareVertexProcessing, presentParams);

device.RenderState.CullMode = Cull.CounterClockwise;

b = (Bitmap)System.Drawing.Image.FromFile("wall.png");

brickwall = new Texture(device, b, 0, Pool.Managed);

cracks = new Texture[4];

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

{

b = (Bitmap)System.Drawing.Image.FromFile("crack" + i + ".png");

cracks[i] = new Texture(device, b, 0, Pool.Managed);

}

}

public void CameraPositioning()

{

device.Transform.Projection = Matrix.PerspectiveFovLH((float)Math.PI / 4, (float)this.Width / this.Height, 1f, 120f);

device.RenderState.Lighting = true;

device.Lights[0].Type = LightType.Directional;

device.Lights[0].Diffuse = Color.Red;

device.Lights[0].Direction = new Vector3(0, 0, -1);

device.Lights[0].Enabled = true;

device.Lights[1].Type = LightType.Point;

device.Lights[1].Position = new Vector3(-50, -40, 1);

device.Lights[1].Range = 1f;

device.Lights[1].Diffuse = Color.Blue;

device.Lights[1].Ambient = Color.Blue;

device.Lights[1].Attenuation0 = 0f;

device.Lights[1].Attenuation1 = 0f;

device.Lights[1].Attenuation2 = 0.5f;

device.Lights[1].Enabled = true;

device.Lights[2].Type = LightType.Spot;

device.Lights[2].Position = new Vector3(50, 40, 5);

device.Lights[2].Direction = new Vector3(0, 0, -1);

device.Lights[2].Range = 10f;

device.Lights[2].Diffuse = Color.Green;

device.Lights[2].Ambient = Color.Green;

device.Lights[2].Attenuation0 = 0f;

device.Lights[2].Attenuation1 = 0.01f;

device.Lights[2].Attenuation2 = 0f;

device.Lights[2].InnerConeAngle = 30f;

device.Lights[2].OuterConeAngle = 30f;

device.Lights[2].Enabled = true;

}

public void VertexDeclaration()

{

brickwall\_vb = new VertexBuffer(typeof(CustomVertex.PositionNormalTextured), 4, device, Usage.Dynamic | Usage.WriteOnly, CustomVertex.PositionNormalTextured.Format, Pool.Default);

//wall

brickwall\_vertices = new CustomVertex.PositionNormalTextured[4];

brickwall\_vertices[0] = new CustomVertex.PositionNormalTextured(-50f, -40f, 0f, 0f, 0f, 1f, 1f, 1f);

brickwall\_vertices[1] = new CustomVertex.PositionNormalTextured(-50f, 40f, 0f, 0f, 0f, 1f, 1f, 0f);

brickwall\_vertices[2] = new CustomVertex.PositionNormalTextured(50f, 40f, 0f, 0f, 0f, 1f, 0f, 0f);

brickwall\_vertices[3] = new CustomVertex.PositionNormalTextured(50f, -40f, 0f, 0f, 0f, 1f, 0f, 1f);

cracks0\_vb = new VertexBuffer(typeof(CustomVertex.PositionNormalTextured), 4, device, Usage.Dynamic | Usage.WriteOnly, CustomVertex.PositionNormalTextured.Format, Pool.Default);

//crack0

cracks0\_vertices = new CustomVertex.PositionNormalTextured[4];

cracks0\_vertices[0] = new CustomVertex.PositionNormalTextured(-10f, 0f, 0f, 0f, 0f, 1f, 1f, 1f);

cracks0\_vertices[1] = new CustomVertex.PositionNormalTextured(-10f, 40f, 0f, 0f, 0f, 1f, 1f, 0f);

cracks0\_vertices[2] = new CustomVertex.PositionNormalTextured(10f, 40f, 0f, 0f, 0f, 1f, 0f, 0f);

cracks0\_vertices[3] = new CustomVertex.PositionNormalTextured(10f, 0f, 0f, 0f, 0f, 1f, 0f, 1f);

cracks1\_vb = new VertexBuffer(typeof(CustomVertex.PositionNormalTextured), 4, device, Usage.Dynamic | Usage.WriteOnly, CustomVertex.PositionNormalTextured.Format, Pool.Default);

//crack1

cracks1\_vertices = new CustomVertex.PositionNormalTextured[4];

cracks1\_vertices[0] = new CustomVertex.PositionNormalTextured(-50f, -40f, 0f, 0f, 0f, 1f, 1f, 1f);

cracks1\_vertices[1] = new CustomVertex.PositionNormalTextured(-50f, 0f, 0f, 0f, 0f, 1f, 1f, 0f);

cracks1\_vertices[2] = new CustomVertex.PositionNormalTextured(10f, 0f, 0f, 0f, 0f, 1f, 0f, 0f);

cracks1\_vertices[3] = new CustomVertex.PositionNormalTextured(10f, -40f, 0f, 0f, 0f, 1f, 0f, 1f);

cracks2\_vb = new VertexBuffer(typeof(CustomVertex.PositionNormalTextured), 4, device, Usage.Dynamic | Usage.WriteOnly, CustomVertex.PositionNormalTextured.Format, Pool.Default);

//crack2

cracks2\_vertices = new CustomVertex.PositionNormalTextured[4];

cracks2\_vertices[0] = new CustomVertex.PositionNormalTextured(10f, -40f, 0f, 0f, 0f, 1f, 1f, 1f);

cracks2\_vertices[1] = new CustomVertex.PositionNormalTextured(10f, 40f, 0f, 0f, 0f, 1f, 1f, 0f);

cracks2\_vertices[2] = new CustomVertex.PositionNormalTextured(50f, 40f, 0f, 0f, 0f, 1f, 0f, 0f);

cracks2\_vertices[3] = new CustomVertex.PositionNormalTextured(50f, -40f, 0f, 0f, 0f, 1f, 0f, 1f);

cracks3\_vb = new VertexBuffer(typeof(CustomVertex.PositionNormalTextured), 4, device, Usage.Dynamic | Usage.WriteOnly, CustomVertex.PositionNormalTextured.Format, Pool.Default);

//crack3

cracks3\_vertices = new CustomVertex.PositionNormalTextured[4];

cracks3\_vertices[0] = new CustomVertex.PositionNormalTextured(-50f, 0f, 0f, 0f, 0f, 1f, 1f, 1f);

cracks3\_vertices[1] = new CustomVertex.PositionNormalTextured(-50f, 40f, 0f, 0f, 0f, 1f, 1f, 0f);

cracks3\_vertices[2] = new CustomVertex.PositionNormalTextured(-10f, 40f, 0f, 0f, 0f, 1f, 0f, 0f);

cracks3\_vertices[3] = new CustomVertex.PositionNormalTextured(-10f, 0f, 0f, 0f, 0f, 1f, 0f, 1f);

brickwall\_vb.SetData(brickwall\_vertices, 0, LockFlags.None);

cracks0\_vb.SetData(cracks0\_vertices, 0, LockFlags.None);

cracks1\_vb.SetData(cracks1\_vertices, 0, LockFlags.None);

cracks2\_vb.SetData(cracks2\_vertices, 0, LockFlags.None);

cracks3\_vb.SetData(cracks3\_vertices, 0, LockFlags.None);

ib\_1 = new IndexBuffer(typeof(int), 6, device, Usage.WriteOnly, Pool.Default);

indices = new int[]

{

2, 1, 0,

3, 2, 0

};

ib\_1.SetData(indices, 0, LockFlags.None);

}

protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)

{

device.Clear(ClearFlags.Target, Color.DarkBlue, 1.0f, 0);

device.BeginScene();

device.VertexFormat = CustomVertex.PositionNormalTextured.Format;

Material M = new Material();

M.Diffuse = Color.White;

//M.Emissive = Color.White;

device.Material = M;

device.Transform.View = Matrix.LookAtLH(new Vector3((float)(100 \* Math.Sin(angle)), 0f, (float)(100 \* Math.Cos(angle))),

new Vector3(0, 0, 0),

new Vector3(0, 1, 0));

//установка вершин и индексов, показывающих как из них построить поверхность

device.SetStreamSource(0, brickwall\_vb, 0);

device.Indices = ib\_1;

device.SetTexture(0, brickwall);

device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, 4, 0, 2);

device.SetStreamSource(0, cracks0\_vb, 0);

device.SetTexture(0, cracks[0]);

device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, 4, 0, 2);

device.SetStreamSource(0, cracks1\_vb, 0);

device.SetTexture(0, cracks[1]);

device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, 4, 0, 2);

device.SetStreamSource(0, cracks2\_vb, 0);

device.SetTexture(0, cracks[2]);

device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, 4, 0, 2);

device.SetStreamSource(0, cracks3\_vb, 0);

device.SetTexture(0, cracks[3]);

device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, 4, 0, 2);

device.EndScene();

device.Present();

angle += 0.01f;

this.Invalidate();

}

}

}