**Practical No. 6:**

**Aim: Programmable Diffuse Lightning in a traingle using Direct3D 11**

* **Transforms.Projection** property: Retrieves or sets the projection transformation Matrix.
* **Matrix.PerspectiveFovLH** method: Builds a left-handed perspective projection matrix based on a field of view.
* **Input arguments to PerspectiveFovLH** are as follows.

//Values provided below are examples

**1) Field of view in radians: pi/4.**

**2) Aspect ratio, or view-space height divided by width: 1, for a square window.**

**3) Near clipping plane distance: 1 unit.**

**4) Far clipping plane distance: 100 units.**

Syntax Example

device.Transform.Projection = Matrix.PerspectiveFovLH((float)Math.PI / 4, 1.0f, 1.0f, 100.0f );

* **Transforms.View**property: Retrieves or sets the view transformation matrix.
* **Matrix.LookAtLH**method: Builds a left-handed look-at matrix.
* **The three input vectors** represent the following, respectively:

//Values provided below are just for example

**1)** The eye point: [0, 3, -5].

**2)** The camera look-at target: the origin [0, 0, 0].

**3)** The current world's up-direction: usually [0, 1, 0].

Syntax Example

device.Transform.View = Microsoft.DirectX.Matrix.LookAtLH(

new Vector3(0.0f, 3.0f, -5.0f), new Vector3(0.0f, 0.0f, 0.0f), new Vector3(0.0f, 1.0f, 0.0f));

* **device.RenderState**property: Retrieves a render-state value for a device.
* **CustomVertex.PositionTextured**structure: Describes a custom vertex format structure that contains position and one set of texture coordinates.
* **Texture.FromBitmap(Device,Bitmap,Usage,Pool)** method: Creates a texture resource from a bitmap.
* **Device.SetTexture**method: Assigns a texture to a device stage.
* **Light** class: Defines a set of lighting properties.
* **Properties are:**

**1) Light.Type:** Retrieves or sets the type of the light source.

**2) Light.Diffuse:** Retrieves or sets the diffuse color emitted by the light.

**3) Light.Enabled:** Retrieves or sets a Boolean value that enables or disables a set of lighting parameters within a device.

**4) Light.Direction:** Retrieves or sets the direction in which the light is pointing in world space, as specified by a Vector3 structure.

* **Vector3:** Describes and manipulates a vector in three-dimensional (3-D) space.

1. After **completing the steps from the initialization file(“base setup.pdf”)**, now **open**

**“Form1.cs” file** in your project, and **code** the part where it is commented as **//OUR CODE**

***-----Form1.cs------***

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using Microsoft.DirectX; //OUR CODE

using Microsoft.DirectX.Direct3D; //OUR CODE

namespace p7

{

public partial class Form1 :Form

{

Microsoft.DirectX.Direct3D.Device device; //OUR CODE

public Form1()

{

InitializeComponent();

InitDevice(); //OUR CODE

}

private void InitDevice() //OUR CODE

{

PresentParameters pp = new PresentParameters(); pp.Windowed = true;

pp.SwapEffect = SwapEffect.Discard;

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

device.RenderState.Lighting = false;

device.Transform.Projection = Matrix.PerspectiveFovLH(3.14f / 4, device.Viewport.Width / device.Viewport.Height, 1f, 100f);

device.Transform.View = Matrix.LookAtLH(new Vector3(0, 0, 10), new Vector3(), new Vector3(0, 1, 0));

device.RenderState.Lighting = true;

device.Lights[0].Type = LightType.Directional; device.Lights[0].Diffuse = Color.Plum;

device.Lights[0].Direction = newVector3(0.8f, 0, -1);

device.Lights[0].Enabled = true;

}

private void Render() //OUR CODE

{

CustomVertex.PositionNormalColored[] vertexes = new

CustomVertex.PositionNormalColored[3]; //3 here is the number of vectors you've defined

//triangle1

vertexes[0] = new CustomVertex.PositionNormalColored(new Vector3(0, 1, 1), new Vector3(1, 0, 1), Color.Red.ToArgb());

vertexes[1] = new CustomVertex.PositionNormalColored(new Vector3(-1, -1,

1), new Vector3(1, 0, 1), Color.Red.ToArgb());

vertexes[2] = new CustomVertex.PositionNormalColored(new Vector3(1, -1,

1), new Vector3(-1, 0, 1), Color.Red.ToArgb());

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

device.BeginScene();

device.VertexFormat=CustomVertex.PositionNormalColored.Format; device.DrawUserPrimitives(PrimitiveType.TriangleList, vertexes.Length / 3, vertexes);

//in the above line(vertexes.Length=3, so 3/3=1) hence, 1 for single triangle & 2 for double triangle

device.EndScene();

device.Present();

}

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Render(); //OUR CODE

}

}

}

1. Click the **Start button** to **run file**> (you’ll see the following **OUTPUT** of the window with the color you specified).

