Министерство науки и высшего образования Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №4

по дисциплине

«Компьютерная графика»

Факультет прикладной математики и информатики

Группа ПМ-63

Студенты Майер В.А.

Преподаватели Задорожный А.Г.

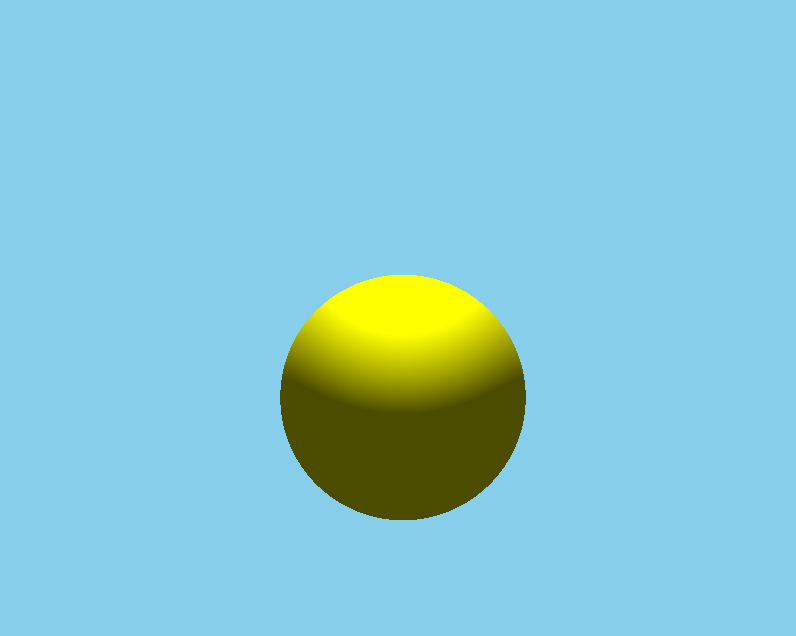
Новосибирск 2019

1. Цель работы

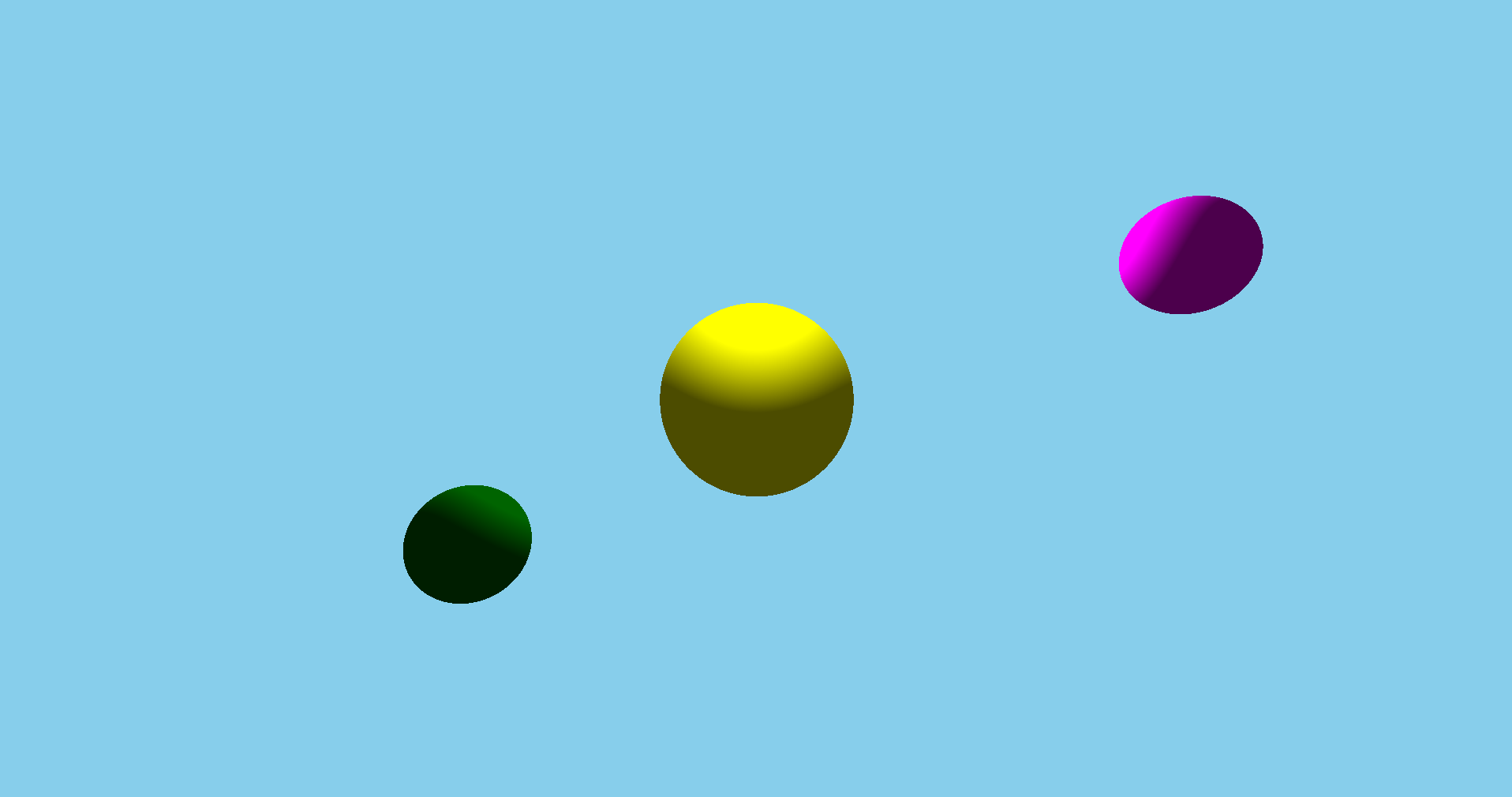
Ознакомится с основными аспектами метода трассировки лучей.

1. **Тесты**

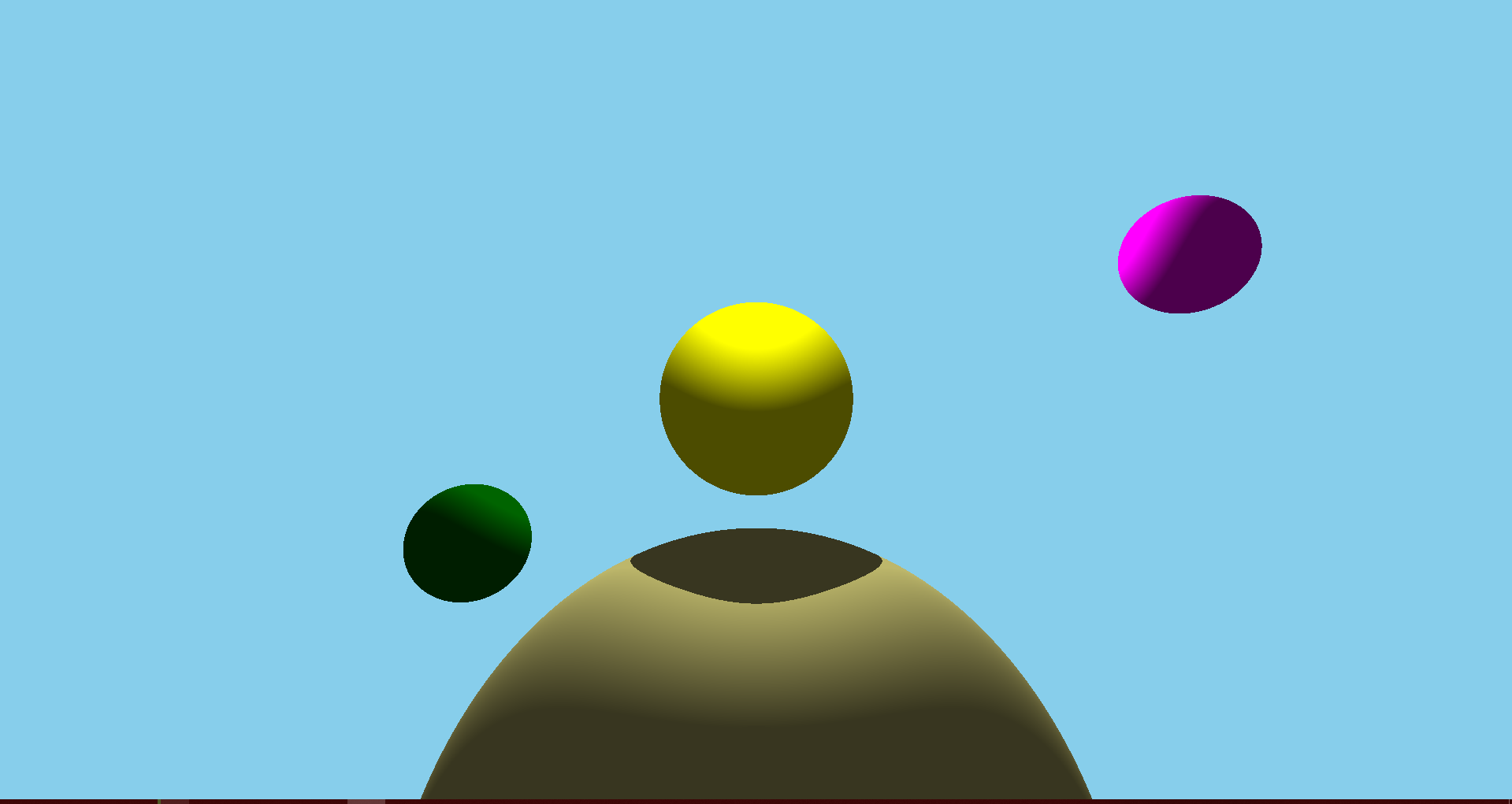
Один объект



Несколько объектов

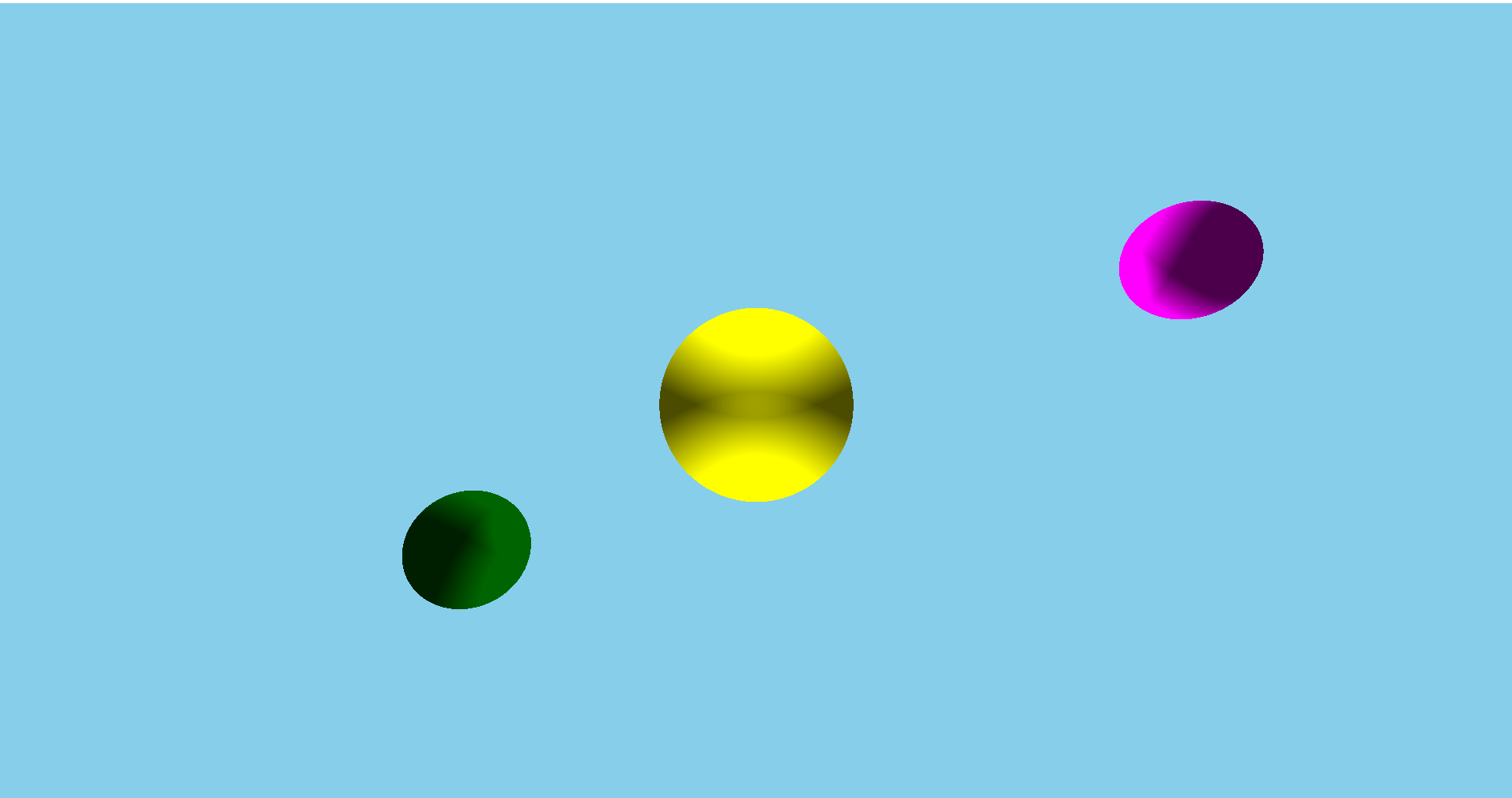


Тени

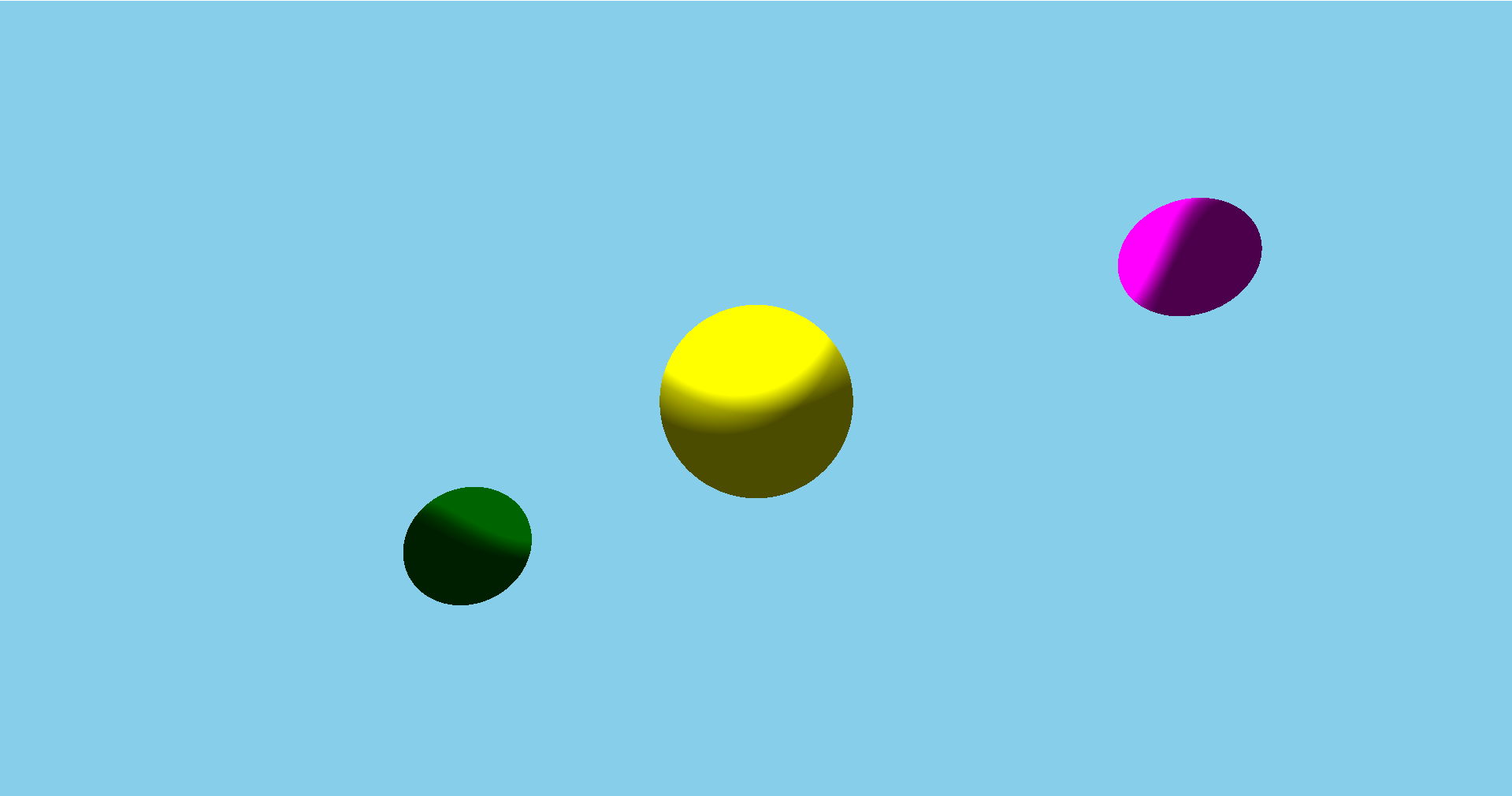


Несколько источников света

С разных сторон



С одной стороны



1. **Текст программы**

using System;

using System.Threading;

using System.IO;

using System.Text;

using System.Collections.Generic;

using System.Drawing;

using System.Drawing.Imaging;

using OpenTK;

using OpenTK.Graphics;

using OpenTK.Graphics.OpenGL;

using OpenTK.Input;

namespace Example

{

class Object

{

public enum TypeObject

{

Sphere,

Box

}

//Текустура

private TypeObject type;

private float r;

private Vector3 worldCoordinat;

private Vector2 angele;

public int maxN;

public Color color;

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="Pos">Позиция объекта</param>

/// <param name="Anegle">Угол поворота</param>

/// <param name="Color">Цвет объекта</param>

/// <param name="MaxLightReflection">Макимальный количество отражения света</param>

/// <param name="TypeObject">Тип объекта</param>

/// <param name="radius">Радиус объекта</param>

public Object(Vector3 Pos, Vector2 Anegle, Color Color, int MaxLightReflection, TypeObject TypeObject, float radius)

{

worldCoordinat = Pos;

angele = Anegle;

maxN = MaxLightReflection;

color = Color;

r = radius;

}

/// <summary>

/// Проверяет пересекает ли луч, ланный объект

/// </summary>

/// <param name="e"> Начальнаня точка </param>

/// <param name="d"> Направление луча </param>

/// <returns> -1 если не найден Меньше 1 если не видно на экране, </returns>

public double CheckTrack(Vector3 e, Vector3 d)

{

double t = 0;

e -= worldCoordinat;

switch (type)

{

case TypeObject.Sphere:

{

double a = Oper.Scolar(d, d);

double b = 2 \* Oper.Scolar(e, d);

double c = Oper.Scolar(e, e) - Math.Pow(r, 2);

double D = Math.Pow(b, 2) - 4 \* a \* c;

if (D < 0) t = -1;

else t = (-b + Math.Sqrt(D)) / (2.0 \* a);

}

break;

case TypeObject.Box:

break;

default:

break;

}

return t;

}

/// <summary>

/// Поиск точки пересечения

/// </summary>

/// <param name="e"> Начальная точка </param>

/// <param name="d"> Направление луча </param>

/// <param name="t"> Значение </param>

/// <returns> Точка пресечения объекта с лучем</returns>

public Vector3 FindPoint(Vector3 e, Vector3 d, double t)

{

return e + Oper.Mult(d,(float)t);

}

public Vector3 GetNorm(Vector3 Point)

{

switch (type)

{

case TypeObject.Sphere:

return new Vector3(worldCoordinat.X - Point.X, worldCoordinat.Y - Point.Y, Point.Z - worldCoordinat.Z).Normalized();

case TypeObject.Box:

return new Vector3();

default:

return new Vector3();

}

}

}

class Light

{

public enum Type

{

Point,

Projector

}

private Vector3 pos;

private Vector3 direction;

private Type type;

private double CoefZatih;

private double I0;

private Func<double, double> LightFunc;

/// <summary>

/// Срздание источника света

/// </summary>

/// <param name="Pos"> Положение </param>

/// <param name="Direction"> Направление </param>

/// <param name="Type"> Тип </param>

/// <param name="I0"> Начальная интенисивность </param>

/// <param name="coef"> Коэфициент затухания </param>

public Light(Vector3 Pos, Vector3 Direction, Type Type, double I0, double coef)

{

pos = Pos;

direction = Direction;

type = Type;

CoefZatih = coef;

this.I0 = I0;

}

public bool CheckCollision(Vector3 Point, List<Object> objects, int l)

{

double t = objects[l].CheckTrack(pos, (Point - pos));

double t1;

for (int i = 0; i < objects.Count; i++)

{

t1 = objects[i].CheckTrack(pos, (Point - pos));

if (i != l && t1 > 0 && t1 < t)

return false;

}

return true;

}

/// <summary>

/// Интенсивность света в точке

/// </summary>

/// <param name="Point"> Точка на которую падает свет </param>

/// <param name="obj"> Объекут, чтобоы посчитать норму в этой точке </param>

/// <returns></returns>

public float Intensivnost(Vector3 Point, Object obj)

{

switch (type)

{

case Type.Point:

return (float)Math.Max(Oper.Scolar(obj.GetNorm(Point), Point - pos) / Oper.Norm3D(Point - pos) / Oper.Norm3D(obj.GetNorm(Point)), 0) \* ((float)I0 - (Point - pos).Length \* (float)CoefZatih);

case Type.Projector:

return (float)Math.Max(Oper.Scolar(obj.GetNorm(Point), direction) / Oper.Norm3D(obj.GetNorm(Point)) / Oper.Norm3D(direction), 0) \* ((float)I0 - (Point - pos).Length \* (float)CoefZatih);

default:

return 0f;

}

}

};

class Camera

{

private Vector3 e; //Полопжение камеры

private Vector3 d; //Направление камеры

private int width; //Ширина

private int height; //Высота

private double fov; //Угол обзора

private List<Object> objects; //Объекты

private List<Light> lights; //Источники света

public Bitmap bm;

/// <summary>

/// Создание камеры

/// </summary>

/// <param name="pos"> Позиция камеры </param>

/// <param name="D"> Направления камеры </param>

/// <param name="Width"> Ширина экрана </param>

/// <param name="Height"> Высота экрана </param>

public Camera(Vector3 pos, Vector3 D, int Width, int Height, double Fov)

{

e = pos;

d = D;

width = Width;

height = Height;

fov = Fov;

bm = new Bitmap(width, height);

}

/// <summary>

/// Задаем Объекты

/// </summary>

/// <param name="Objects"> Список всех объектов на сцене (Включая куб видимости,

/// если его нет то будет черный цвет в месах, где нет пересечения) </param>

public void SetObjects(List<Object> Objects)

{

objects = Objects;

}

/// <summary>

/// Задаем источники света

/// </summary>

/// <param name="Lights"> Список всех источников света на сцене </param>

public void SetLights(List<Light> Lights)

{

lights = Lights;

}

public void Print(float ambet, float reflaction)

{

for (int j = 0; j < height; j++)

{

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

{

double t;

double mint = -1.0;

int nobj = -1;

float x = (float)((2 \* (i + 0.5) / (float)width - 1) \* Math.Tan(fov / 2.0) \* width / (float)height);

float y = (float)(-(2 \* (j + 0.5) / (float)height - 1) \* Math.Tan(fov / 2.0));

Vector3 dir = new Vector3(x, y, -1).Normalized();

for (int k = 0; k < objects.Count; k++)

{

t = objects[k].CheckTrack(e, dir);

if (t > 0 && (t < mint || mint < 0))

{

mint = t;

nobj = k;

}

}

if (nobj == -1)

bm.SetPixel(i, j, Color.SkyBlue);

else

{

double intens = 0;

for (int l = 0; l < lights.Count; l++)

{

if (lights[l].CheckCollision(e + Oper.Mult(dir, (float)mint), objects, nobj)) //Тени

intens += lights[l].Intensivnost((e + Oper.Mult(dir, (float)mint)), objects[nobj]);

}

intens = Math.Min(intens + ambet, 1);

bm.SetPixel(i, j, Oper.Mult(objects[nobj].color, (float)intens));

}

}

}

}

}

partial class MyApplication

{

//Иноформирует активное акно или нет

static bool activ = true;

//Загрузка текстуры по патчу file

public static int LoadTexture(string file)

{

Bitmap bitmap = new Bitmap(file);

int tex;

GL.Hint(HintTarget.PerspectiveCorrectionHint, HintMode.Nicest);

GL.GenTextures(1, out tex);

GL.BindTexture(TextureTarget.Texture2D, tex);

BitmapData data = bitmap.LockBits(new System.Drawing.Rectangle(0, 0, bitmap.Width, bitmap.Height),

ImageLockMode.ReadOnly, System.Drawing.Imaging.PixelFormat.Format32bppArgb);

GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba, data.Width, data.Height, 0,

OpenTK.Graphics.OpenGL.PixelFormat.Bgra, PixelType.UnsignedByte, data.Scan0);

bitmap.UnlockBits(data);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter, (int)TextureMinFilter.Linear);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter, (int)TextureMagFilter.Linear);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapS, (int)TextureWrapMode.Repeat);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapT, (int)TextureWrapMode.Repeat);

return tex;

}

[STAThread]

public static void Main()

{

int LightReflaction = 3;

using (var game = new GameWindow())

{

game.Title = "Lab3";

game.WindowState = WindowState.Maximized;

game.WindowBorder = WindowBorder.Fixed | WindowBorder.Hidden;

game.CursorVisible = false;

BitmapData data = new BitmapData();

bool pof = true;

int textureID = LoadTexture("C:/Users/PM65M/OneDrive/Desktop/Graf2.jpg");

List<Object> objects = new List<Object>();

List<Light> lights = new List<Light>();

Camera camera = new Camera(new Vector3(), new Vector3(1,0, 0), game.Width, game.Height, 70.0 / 180.0 \* Math.PI);

#region Создание объектов

//objects.Add(new Object(new Vector3(0, -8, -30), new Vector2(0, 0), Color.Red, LightReflaction, Object.TypeObject.Sphere, 2));

objects.Add(new Object(new Vector3(0, 0, -30), new Vector2(0, 0), Color.Yellow, LightReflaction, Object.TypeObject.Sphere, 5));

// objects.Add(new Object(new Vector3(0, -2, -40), new Vector2(0, 0), Color.Blue, LightReflaction, Object.TypeObject.Sphere, 1));

objects.Add(new Object(new Vector3(-20, 10, -40), new Vector2(0, 0), Color.DarkGreen, LightReflaction, Object.TypeObject.Sphere, 4));

objects.Add(new Object(new Vector3(30, -10, -40), new Vector2(0, 0), Color.Magenta, LightReflaction, Object.TypeObject.Sphere, 4));

objects.Add(new Object(new Vector3(0, 20, -25), new Vector2(0, 0), Color.DarkKhaki, LightReflaction, Object.TypeObject.Sphere, 14));

//objects.Add(new Object(new Vector3(0, 0, 0), new Vector2(0, 0), Color.SkyBlue, 0, Object.TypeObject.Box, 1000 ));

#endregion

#region Создание света

lights.Add(new Light(new Vector3(0, -30, -30), new Vector3(0, 0, 0), Light.Type.Point, 1, 0));

#endregion

game.Load += (sender, e) =>

{

game.VSync = VSyncMode.On;

GL.LineWidth(3);

camera.SetLights(lights);

camera.SetObjects(objects);

camera.Print(0.3f, 0.0f);

data = camera.bm.LockBits(new Rectangle(0, 0, camera.bm.Width, camera.bm.Height), ImageLockMode.ReadOnly, camera.bm.PixelFormat);

};

game.Resize += (sender, e) =>

{

GL.Viewport(0, 0, game.Width, game.Height);

};

game.KeyDown += (sender, e) =>

{

if (e.Key == Key.Escape)

game.Close();

};

game.UpdateFrame += (sender, e) =>

{

};

game.RenderFrame += (sender, e) =>

{

// render graphics

GL.Clear(ClearBufferMask.DepthBufferBit | ClearBufferMask.ColorBufferBit | ClearBufferMask.StencilBufferBit);

GL.DrawPixels(camera.bm.Width, camera.bm.Height, OpenTK.Graphics.OpenGL.PixelFormat.Bgra, PixelType.UnsignedByte, data.Scan0);

game.SwapBuffers();

};

//60 кадров в сек

game.Run(10.0);

}

}

}

class Oper

{

//Норма по осям X, Z

public static double Norm2D(Vector3 p)

{

return Math.Sqrt(Math.Pow(p.X, 2) + Math.Pow(p.Z, 2));

}

public static double Norm3D(Vector3 p)

{

return Math.Sqrt(Math.Pow(p.X, 2) + Math.Pow(p.Y, 2) + Math.Pow(p.Z, 2));

}

public static double Scolar(Vector3 a, Vector3 b)

{

return a.X \* b.X + a.Y \* b.Y + a.Z \* b.Z;

}

public static Vector3 Normal(Vector3 a, Vector3 b, Vector3 c)

{

b = b - a;

c = a - c;

return new Vector3(b.Y \* c.Z - b.Z \* c.Y, b.Z \* c.X - b.X \* c.Z, b.X \* c.Y - b.Y \* c.X);

}

public static Vector3 Mult(Vector3 a, float b)

{

return new Vector3(a.X \* b, a.Y \* b, a.Z \* b);

}

public static Color Mult(Color a, float b)

{

if (b > 1)

b = 1;

return Color.FromArgb((int)(a.R \* b), (int)(a.G \* b), (int)(a.B \* b));

}

}

}