MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE

" IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Victor Porev

Computer Graphics Programming

Laboratory Work 6

Ray Tracing

Kulubecioglu Mehmet

IM-14 FIOT

Class Number: 12

Kyiv

IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE

2025

2. Project Purpose and Scope

The objective of this lab work is to understand and apply the ray tracing algorithm to construct photorealistic scenes in 3D computer graphics. Ray tracing simulates real-world light behaviors such as reflection, refraction, and shadowing to produce highly realistic images. The project covers the following goals:

- ❖ Scene modeling with objects having various surface properties
- ❖ Simulating interactions of light sources with object surfaces
- ❖ Initiating the rendering process based on camera perspective
- ❖ Developing a window-based application using Win32 API
- ❖ Managing colors and materials in RGB space

3. Used Technologies and File Structure

- ❖ **IDE:** Visual Studio 2022
- ❖ **Programming Language:** C++
- ❖ **API:** Windows API (Win32)
- ❖ **Rendering Logic:** Software-based ray tracing (OpenGL not used)
- ❖ Project Files:
  - ➢ **`Lab6_RAYS.cpp`** = Main window and scene selection logic
  - ➢ **`Raytrace.cpp/h`** = Ray tracing functions for rendering scenes
  - ➢ **`Scene.cpp/h`** = Geometry and scene object definitions
  - ➢ **`Sce_normal.cpp/h`** = Mathematical functions for normal vector calculations
  - ➢ **`Transform.cpp/h`** = Transformation utilities
  - ➢ **`Sce_types.h`** = Vector and geometric type declarations
  - ➢ **`resource.h/.rc`** = Menu, icons, and user interface definitions

4. Scene Descriptions and Technical Details

4.1 Pyramid Reflection

- ❖ **Camera Position:** (250, 500, 200)
- ❖ **Light Source:** White light from (-150, 250, 350)
- ❖ **Ground:** Light purple, soft specular
- ❖ **Objects:** Reflective boxes and a centered pyramid

- **Material:** Pyramid has diffuse-dominant surface, no reflections
- **Tech Notes:** Created with **`SetColor_SCE`**, **`SetMaterial_SCE`**, **`AddPyramid_SCE`**, **`AddQuad_SCE`**

4.2 Sphere on the Table

- **Camera Position:** (350, 600, 320)
- **Light Source:** White light from (-150, 400, 600)
- **Objects:** Wooden table and semi-transparent green sphere
- **Floor:** Checkerboard pattern using **`AddQuad_SCE`** and **`CopyObjectAndShift_SCE`**
- **Sphere:** Includes transparency **(`0.5`)** and refraction **(`0`)**
- **Tech Notes:** `AddCylinder_SCE`, `AddSphere_SCE`, **`AddPrisma4_SCE`** used for modeling

4.3 Transparency

- **Camera Position:** (78, 1200, 500)
- **Light Sources:** Two different colored lights: (1, 0.5, 0) and (0, 0.5, 1)
- **Objects:** Large transparent sphere and checkerboard floor
- **Material:** High level of refraction **(`0.5`)** and transparency (`0.5`)

5. Color and Material Settings

Colors and surface properties are defined via **`SetColor_SCE(r, g, b)`** and **`SetMaterial_SCE(diffuse, specular, reflection, transparency, shininess, refraction)`.** Sample usages:

SetColor_SCE(1, 0.7, 0); // Orange-like color

SetMaterial_SCE(0.5, 0.4, 0.1, 0, 20, 0); // Soft specular material

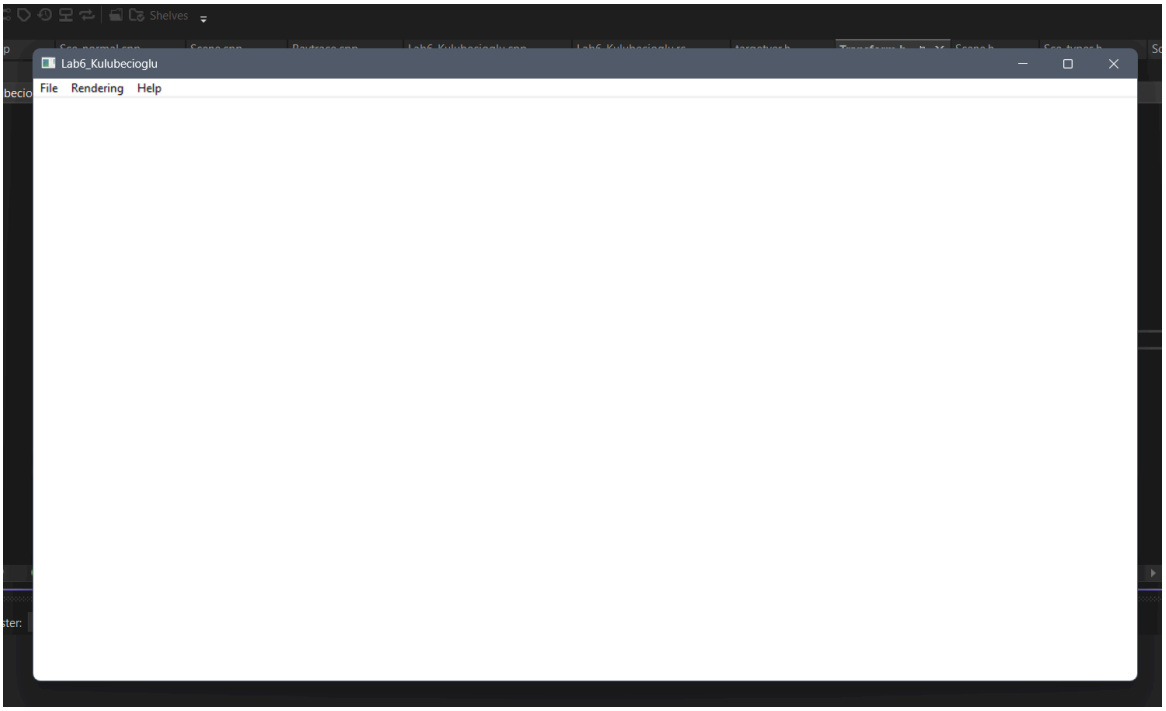
SetMaterial_SCE(0, 0, 0.5, 0.5, 100, 0.5); // Transparent and reflective

Background, floor colors, and light settings can be customized to make the scene more original.

6. Rendering Flow and Application Behavior

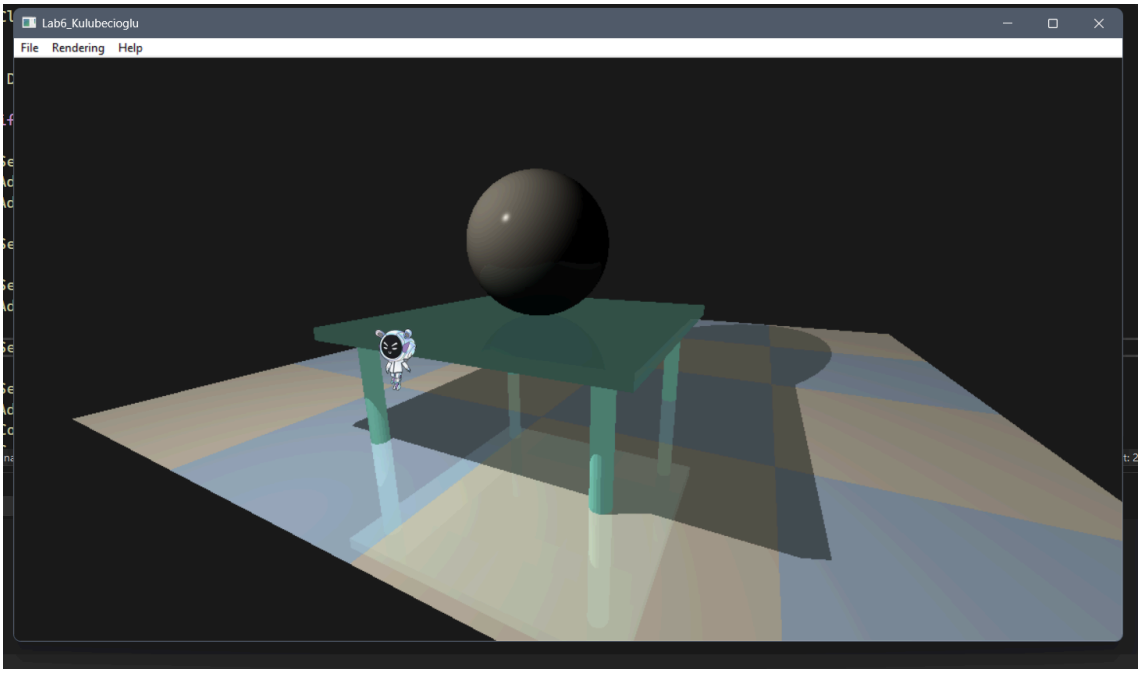
- Application initializes a Win32 window
- Selecting a scene from the menu calls respective **`DrawStudyExample*()`** functions
- Scene is defined and drawn using **`Rendering()`**
- After rendering, the scene is cleared using **`CloseArrays_SCE()`**

7. Screenshots (Placeholders)

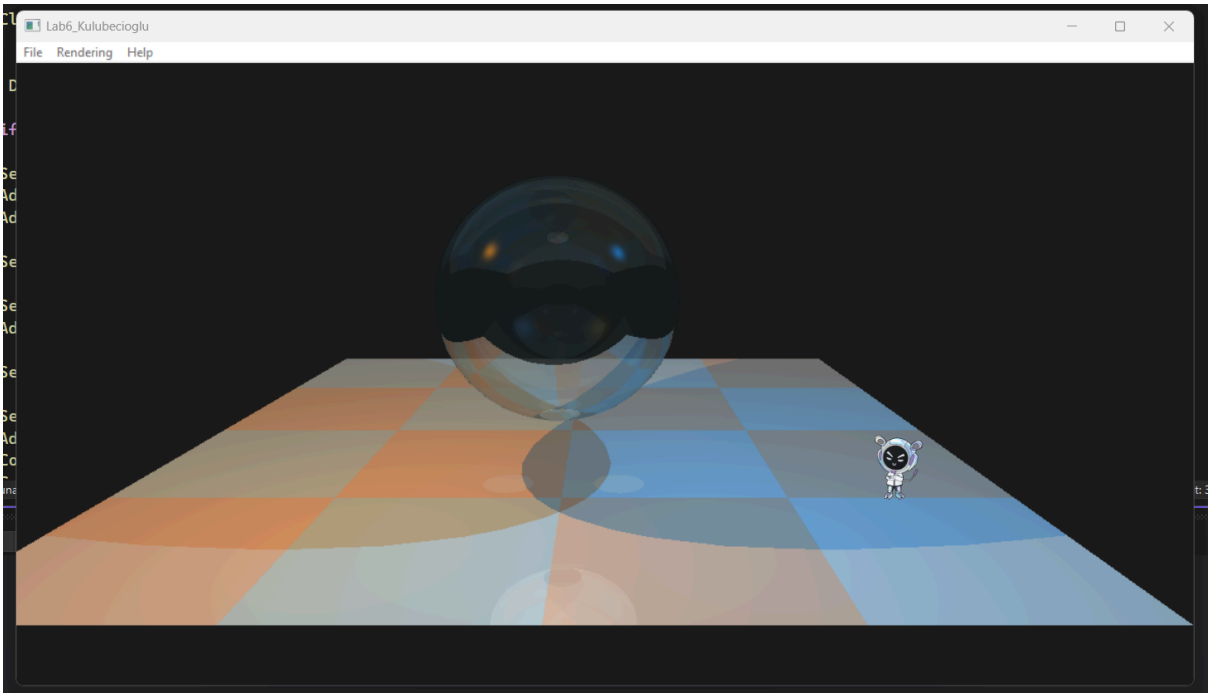


7.1 Pyramid Reflection:

7.2 Sphere on the Table:



7.3 Transparency:



8. Control Questions and Answers

1. **What is ray tracing?**
   Ray tracing is a rendering technique in computer graphics that simulates the physical behavior of light to generate realistic images. It traces the path of rays as they travel from a virtual camera through each pixel on the image plane and into the scene. When a ray intersects an object, the algorithm calculates how the light would reflect, refract, or be absorbed based on material properties. Unlike rasterization, which projects objects onto the screen, ray tracing simulates light transport, making it ideal for achieving realistic effects such as shadows, reflections, transparency, and global illumination.

2. **How are shadows calculated?**
   Shadows are computed by casting a secondary ray, known as a shadow ray, from the point of intersection between the primary viewing ray and the surface of an object. This shadow ray is directed toward each light source in the scene. If this ray intersects another object before reaching the light source, it means that the light is obstructed, and the point is in shadow. The surface's color is then rendered darker or fully black depending on whether soft or hard shadows are implemented. This process creates a realistic perception of depth and spatial relationships in the scene.
3. **How are reflections computed?**
   Reflections are handled by generating reflection rays that bounce off surfaces in the scene. When the primary ray hits a reflective surface, a new ray is calculated based on the angle of incidence, using the surface normal vector to determine the direction of the reflection. This secondary ray is then traced through the scene to detect further interactions, and the resulting color is blended based on the material's reflectivity coefficient. This allows for mirror-like surfaces and complex visual effects like recursive reflections when reflective surfaces face each other.
4. **How is light refraction handled?**
   Light refraction simulates the bending of light as it passes from one medium to another, such as from air into glass. This is computed using Snell's Law, which relates the angles of incidence and refraction to the indices of refraction of the two media. When a ray enters a transparent object, a new refracted ray is generated that changes direction depending on the difference in material densities. Refraction allows for effects such as magnification, distortion, and caustics. The object's transparency and refractive index values determine how the final color is blended with the background.

5. **How is transparency applied?**
   Transparency is implemented by combining the colors of the objects behind a transparent object with the color of the object itself. This is done using alpha blending, where a transparency coefficient defines how much light passes through the surface. A fully transparent object allows all background colors to pass through, while partially transparent objects blend the colors proportionally. Additionally, transparency may be combined with refraction to simulate realistic glass, water, or other translucent materials, accounting for both light passing through and bending within the object.

6. **What is the Whitted model?**
   The Whitted model, introduced by Turner Whitted in 1980, is a foundational algorithm for recursive ray tracing. It combines multiple lighting phenomena such as specular reflection, refraction, shadows, and ambient light into a single framework. In this model, when a ray intersects an object, secondary rays are generated to calculate reflections and refractions, while shadow rays assess direct lighting. Each recursive step adds light contributions until a termination condition is met (like max recursion depth). The Whitted model is ideal for simulating realistic optics but does not handle indirect illumination like global illumination or radiosity.

9. Conclusion

This project demonstrates the practical implementation of ray tracing to render 3D scenes using physical light models in software. Through this application, students gain hands-on experience in manipulating low-level graphics routines, setting up camera views, and modeling object-light interactions. Each scene highlights a different aspect of ray tracing: from reflective surfaces and light bounces to transparency and material blending.

Moreover, the use of pure Win32 API and C++ code emphasizes control over the rendering pipeline without relying on hardware acceleration or high-level graphics engines. The project requires an understanding of mathematical geometry, color theory, and rendering logic. Completing this lab not only reinforces theoretical knowledge but also enhances programming skills necessary for real-time and offline rendering systems. The final output closely resembles photorealistic visuals, underscoring the power and precision of ray tracing algorithms in modern computer graphics.