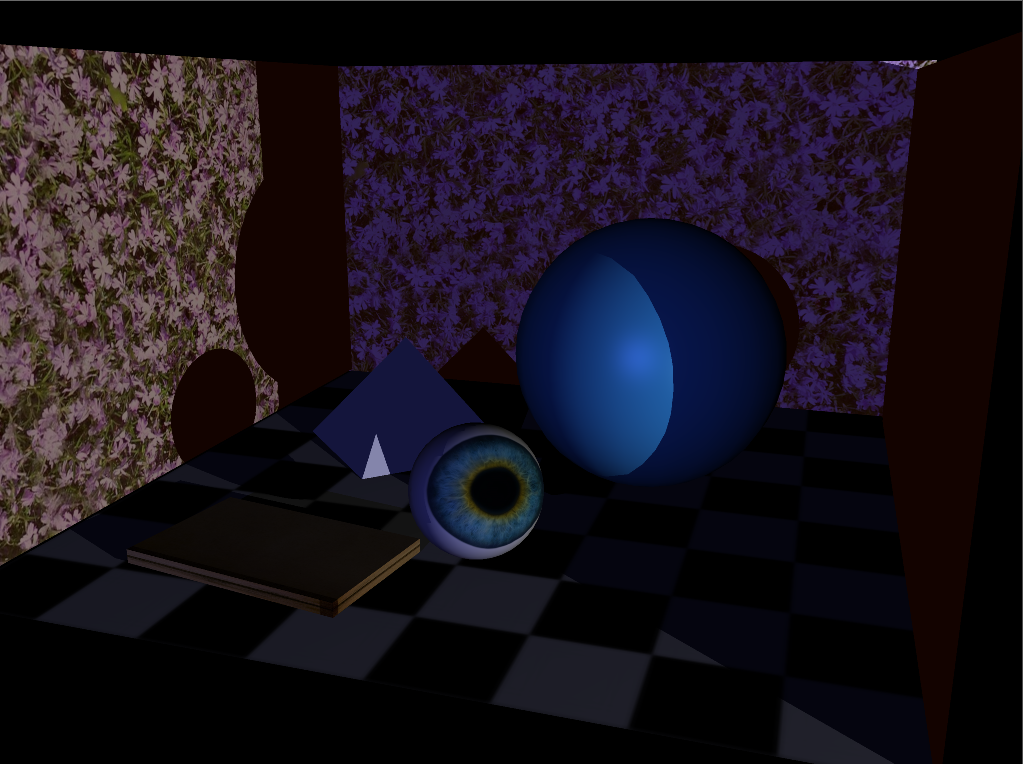
Image Synthesis

Homework 2 – Ray Casting



|  |  |
| --- | --- |
| Name | Dima Ogurtsov |
| ID | 326814910 |
| e-mail | ogurdima@gmail.com |

|  |  |
| --- | --- |
| Name | Yacov Glushkin |
| ID | 326814902 |
| e-mail | yacov.gls@gmail.com |

# Introduction

In this homework we implemented a raycasting algorithm that we learned on lecture.

How to use the plugin:

1. Create polygonal scene
2. Load plugin
3. In script editor type a command “raytrace –w 800 –h 600 –n 10 –s 1”
   1. Parameters are just an example – you can change them of course ☺
   2. –s specifies **samples per dimension** in 2d image: –s 2 means 4 rays per pixel,-s 3 means 9 rays etc. To save your time we recommend use either –s 1 or –s 2 ☹

# Assumptions

The implementation we provided makes the following assumptions:

1. There is at least one object (mesh) in the scene
2. All geometric objects in scene are meshes
3. There are only three light types used in the scene: ambient, point and directional.
4. If an object has a texture it must be monolith – at most one texture per mesh, only one UV set
5. All textures are of FileTexture type (not built in Maya textures but those that you provide using texture file)
6. There are only two types of material – Lambert and Phong.
7. All parameters have positive integer value

# Important Implementation Details

The plugin creates an iff image called "C:/temp/scene.iff" – a resulting image of the ray tracing

The plugin creates txt file called "C:/temp/stat.txt" – statistics of the latest run

Rendering is done using camera with name “cameraShape1”. If no such camera found – using Maya’s active 3d view.

The plugin opens the resulting image in Maya automatically in the end of the run.

During raycasting stage percentage of already computed pixels is printed to the standard output in resolution of 10% (allows you to know that we are not stuck).

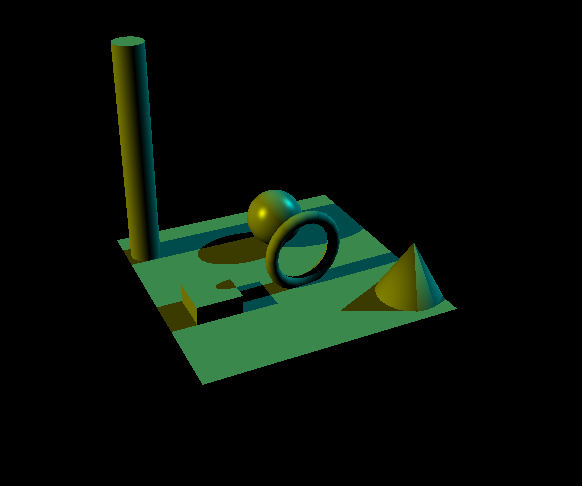
# High Level Overview

High level algorithm is as follows:

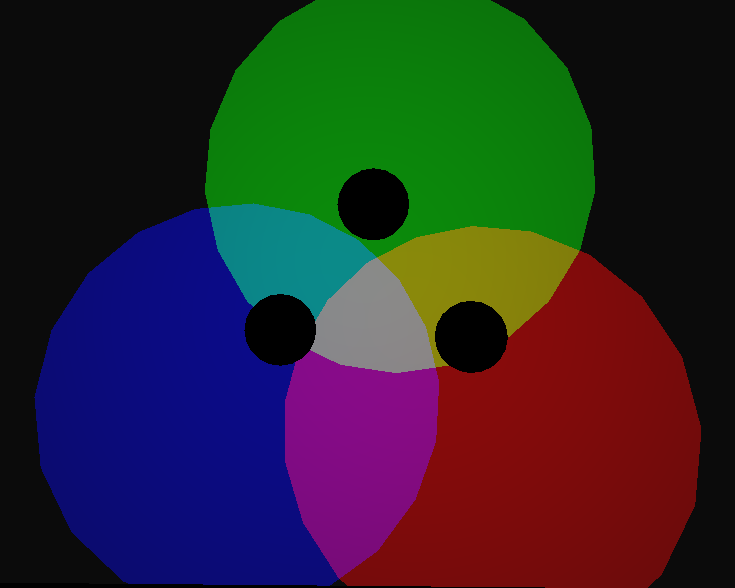
1. Parse user input
2. Collect and store relevant to us scene data (camera data, lights, geometry)
3. Divide 3d scene to voxels in uniform fashion.
   1. Each voxel stores all the polygons that it intersects
4. Follow ray(s) through each pixel using the voxels we computed before, compute closest intersection to camera’s eye.
   1. If the intersection is found and is legit, shoot shadow rays (which also use voxels to traverse the scene) and compute (sub)pixel color.
5. Write the image to a file
6. Write statistics to a file
7. Open the resulting image using Maya API

# Results

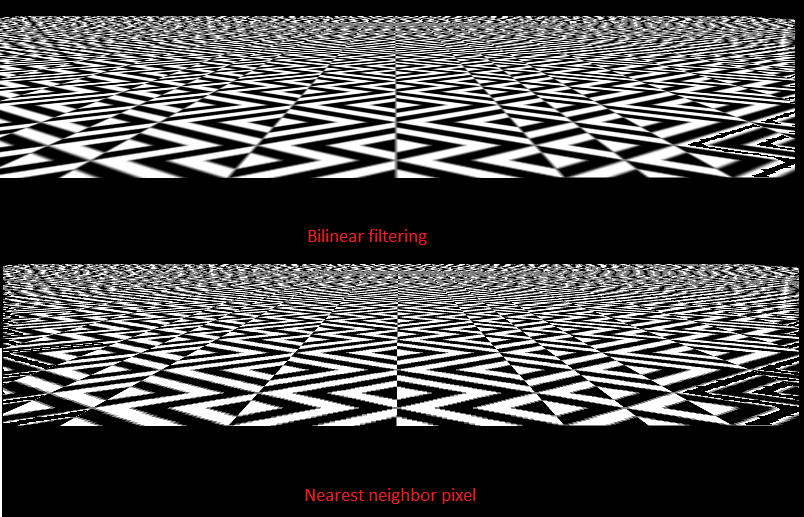
#### Basic rendering



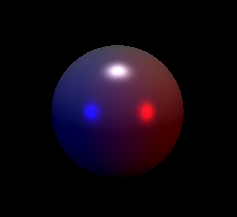
#### Additive lights



#### Bilinear filtering



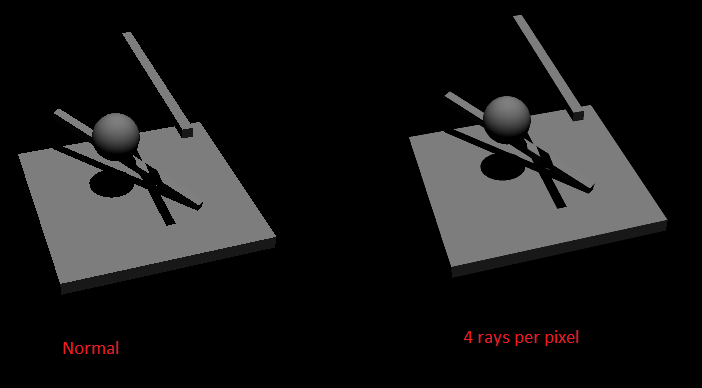
#### Specular highlights



#### Shadows

#### 

#### Super sampling



#### Complex lighting and shadows

