User Documentation

Usage

Run program

Program is executed by calling rt004.exe file in .../src/rt004/bin/Debug/net7.0 [TODO] change the address Or double clicking on the rt004.exe file in your File explorer.

Command line arguments

There is an option to set some parameters from command line as arguments. When executed from command line, then specified arguments have priority over parameters set in config file.

Arguments settable from command line:

Argument	default value	expected value	explanation
config	cwd/config.xml	[path]	Sets path to the file containing scene definition
output	cwd/output.pfm	[path]	Sets path where should be the result saved

Configuration file

The scene is described in a xml file. Which follows couple of rules. The file is automatically deserialized.

The XML File is structured in 3 sections:

- variables - Defines variables used in the next sections of loading as <[VarName]/> unary tag.
- rendererSetting - Defines rendering settings or default values.
- sceneLoader - Defines the scene structure with objects and their properties.

Structure of the file:

```
<DataLoader>
  <variables>
    ...
  </variables>

  <rendererSettings>
    ...
  </rendererSettings>

  <sceneLoader>
    ...
  </sceneLoader>
</DataLoader>
```

RendererSettings

Sets rendering properties and their default values.

tag	default value	value	Description
shadows	true	true / false	Turns on and off shadow computation in the scene.
reflections	true	true / false	Turns on and off reflection computation in the scene

tag	default value	value	Description
refractions	false	true / false	Turns on and off refraction computation in the scene
epsilon	0.001	positive float	Sets the precision of computation in the scene
defaultBackgroundColor	RGBA(0.5, 0.5, 0.5, 1)	tags R, G, B, A	Sets default value of of scene
defaultSolidColor	RGBA(0, 1, 0, 1)	tags R, G, B, A	Sets default base color for all Solid objects scene
defaultCameraWidth	600	unsigned integer	Sets default width of cameras in pixels
defaultCameraHeight	480	unsigned integer	Sets default height of cameras in pixels
defaultSpecularFactor	0	float 0-1	Sets default specular factor for materials. Specular factor effects rays bounced perfectly
defaultDiffuseFactor	1	float 0-1	Sets default diffuse factor for materials. Diffuse factor effects rays bounced in all directions
defaultShininessFactor	2	float	Sets default shininess factor for materials. Shininess factor effects final color intensity
defaultIndexOfRefraction	0.9	float	Sets default index of refraction.
defaultAmbientLightFactor	0.1	float	Sets the intensity factor of ambient light
defaultAmbientLightColor	RGBA(1,1,1,1)	tags R, G, B, A	Default ambient light color in the scene
lightModel	PhongModel	Light calculation model	Defines light model to use for rendering

tag	default value	value	Description
maxReflectionDepth	8	unsigned integer	Max number of hits a recursive ray reflectance can have.
minRayContribution	0.001	float	Minimal contribution of an recursive ray can have to be considered and computed

SceneLoader structure

SceneLoader is separated to two sections:

- ambient light properties - Defines ambient scene light
- sceneObjectLoaders - Defines scene object tree

```
<sceneLoader>
  <ambientLightColor>
    <R>1</R>
    <G>1</G>
    <B>1</B>
    <A>1</A>
  </ambientLightColor>
  <ambientLightIntensity>0.1</ambientLightIntensity>

  <sceneObjectLoaders>

  </sceneObjectLoaders>
</sceneLoader>
```

Some Tags must have a specific type defined. For example SceneObjectLoader must define specific type of the sceneObject.

```
<SceneObjectLoader xsi:type=SphereObjectLoader>
  ...
</SceneObjectLoader>
```

Ambient Light properties

Can defined tags:

tag	default value	expected value	Description
ambientLightColor	RGBA(1,1,1,1)	[tags: R, G, B, A]	Defines ambient color
ambientLightIntensity	0.1	[value]	Defines ambient light intensity

SceneObjectLoaders

Contains SceneObjects and their properties. As a property InnerSceneObjectLoader optionally contains property:"children", which is a list of SceneObjectLoader-s. This recursion forms a structure of scene tree. In the tree, position and rotation of a object is specified relative to the immediate parent SceneObjectLoader position.

Each SceneObject is defined by <SceneObjectLoader xsi:type=[**Loader**]> tag. Where **Loader** can be these types.

Type	Description
InnerSceneObjectLoader	Used as intermediate node in a tree to connect parent node with multiple other SceneObjectLoader-s.
PerspectiveCameraLoader	Perspective Camera to see into the scene.
PointLightLoader	Point light casting light to the scene.
SphereLoader	SceneObjectLoader representing mathematically perfect sphere
PlaneLoader	SceneObjectLoader representing mathematically perfect infinite plane

Universal Properties

Properties universal for all SceneObjects.

property (tag)	default value	expected value	Description
position	XYZ(0,0,0)	[tags x,y,z]	relative position to parent Object or scene origin
rotation	XYZ(0,0,0)	[tags x,y,z]	relative Euler Angles to parent Object or scene origin

Perspective Camera

A camera to render an image of the camera view. The view is oriented along local positive Z axis.

property (tag)	default value	expected value	Description
position	XYZ(0,0,0)	[tags X,Y,Z]	relative position to parent Object or scene origin
rotation	XYZ(0,0,0)	[tags X,Y,Z]	relative Euler Angles to parent Object or scene origin
backgroundColor	RGBA(1,1,1,1)	[tags R,G,B,A]	Scene background color.
fov	90	[float value]	Camera field of view
width	600	[int value]	Width in pixels
height	480	[int value]	Height in pixels

Point Light

Represents light shinning from one point in all directions.

property (tag)	default value	expected value	Description
position	XYZ(0,0,0)	[tags X,Y,Z]	relative position to parent Object or scene origin
rotation	XYZ(0,0,0)	[tags X,Y,Z]	relative Euler Angles to parent Object or scene origin
lightColor	RGBA(1,1,1,1)	[tags R,G,B,A]	Scene background color.
intensity	1	[float value]	The intensity of the light
diffuseFactor	1	[float value]	Defines how much is diffuse part of object lighting affected
specularFactor	1	[float value]	Defines how much is specular part of object lighting affected

Plane

Represents infinite plane in a Scene.

property (tag)	default value	expected value	Description
position	XYZ(0,0,0)	[tags X,Y,Z]	relative position to parent Object or scene origin
rotation	XYZ(0,0,0)	[tags X,Y,Z]	relative Euler Angles to parent Object or scene origin
lightColor	RGBA(1,1,1,1)	[tags R,G,B,A]	Color of the object
intensity	1	[float value]	The intensity of the light
diffuseFactor	1	[float value]	Defines how much is diffuse part of object lighting affected
specularFactor	1	[float value]	Defines how much is specular part of object lighting affected

Sphere

Represents a mathematically perfect sphere (perfectly smooth) in a scene.

property (tag)	default value	expected value	Description
position	XYZ(0,0,0)	[tags X,Y,Z]	relative position to parent Object or scene origin
rotation	XYZ(0,0,0)	[tags X,Y,Z]	relative Euler Angles to parent Object or scene origin
lightColor	RGBA(1,1,1,1)	[tags R,G,B,A]	Scene background color.
diameter	1	[float value]	The diameter of sphere
material	---	[tag material]	Material representing properties of the object

Support Objects

basic types

Values

value type	example	description
integer	0, 1, 2, 10, -5	Integer values

value type	example	description
unsigned integer	0, 1, 2, 10	Only positive integer values and zero
float	0.2, 3, -16.4	Values which can be expressed using floating point
tuple	XYZ(1, 2, 10.3)	It represents connected values([values separated by ,])
RGBA	RGBA(1,1,1,1)	Represents color tuple separated to channels including A for transparency.
tag	tags <value1>...</value1> <value2>...</value2>....	describes, what subtags must be in side of a variable tag
subtag	subtags <value1>...</value1> <value2>...</value2>....	subtags are tags inside of another tag

Light calculation

There is a possibility to change type of light calculation. But right now there is only Phong type implemented and there isn't implemented the change.

Light calculation	Description
PhongModel	Default calculation type using Blinn-Phong light calculation model.

Material

Defines properties of an object depending on Light calculation type. They effect the light hitting the object.

property (tag)	default value	expected value	Description
position	XYZ(0,0,0)	[tags X,Y,Z]	relative position to parent Object or scene origin
rotation	XYZ(0,0,0)	[tags X,Y,Z]	relative Euler Angles to parent Object or scene origin
baseColor	----	[tag texture]	Color of the object

property (tag)	default value	expected value	Description
specularTexture	----	[tag texture]	texture defining specular property at the object. The color values should be in range 0-1.
shininessTexture	----	[tag texture]	texture defining shininess property at the object. The color values should be in range 0-1.
diffuseTexture	----	[tag texture]	texture defining diffuse property at the object. The color values should be in range 0-1.
transparencyTexture	----	[tag texture]	texture defining transparency property at the object. The color values should be in range 0-1.

Textures

Textures represents 2d information with U,V axies. Texture types represents color or just value information.

As a remainder types as used in conjunction of initial tag: <[property name] xsi:type=[type]> ....

Texture type	description
MonochromeUniformTextureLoader	Represents just one float value on the whole texture space.
UniformTextureLoader	Represents one RGBA value on the whole texture space

Variables

Variables can be defined as text (usually sections of xml structure) which is exactly copied. Replacing every instance of <variableName/> outside <variables> tag section.

Variable definition

The variable definitions must be inside a tag "<variables>" in a following structure. And there can be only one variable section. Variable names are not limited, except they **can not** be named the same as any tagname even for.

```
<DataLoader>

  <variables>
    <variable1Name>
      [ variable1 contents ]
    </variable1Name>
```

```
    <variable2Name>
      [ variable2 contents ]
    </variable2Name>
    ...

  </variables>
  ...

<DataLoader>
```

Variable usage

Variables can be defined as text (usually sections of xml structure) which is exactly copied. Replacing every instance of <variableName/> outside <variables> tag section.

For example:

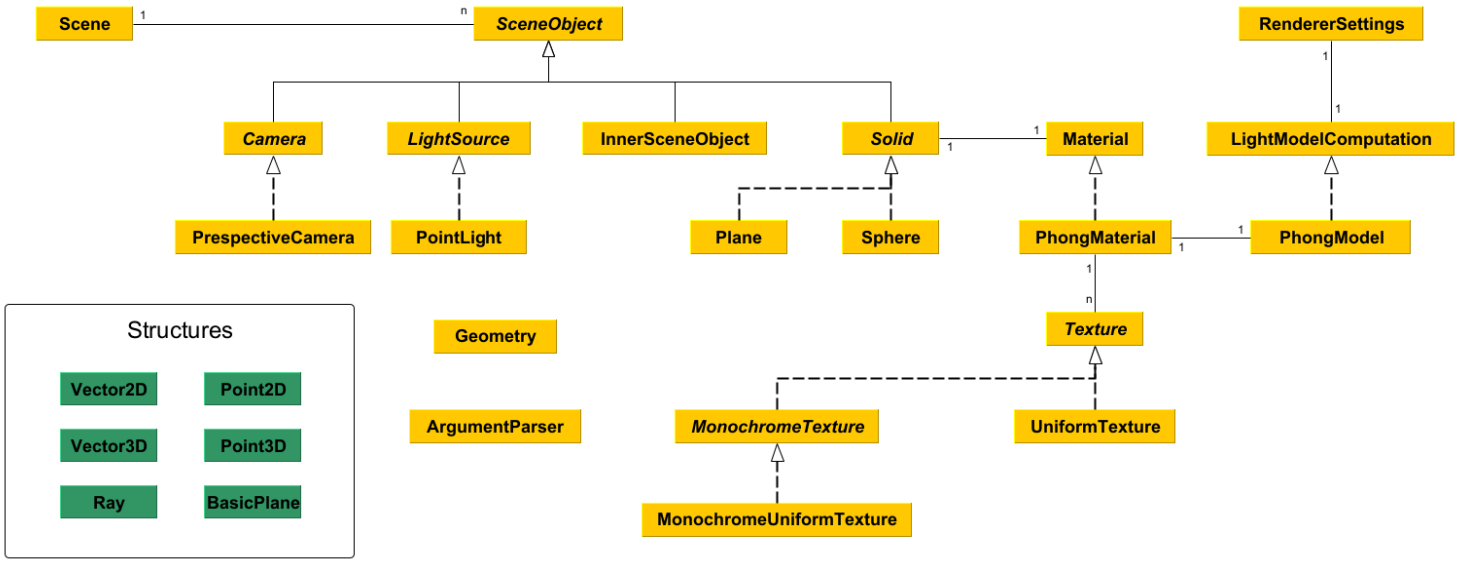
```
<DataLoader>
  <variables>
    <MyVariable>
      <SceneObjectLoader xsi:type="SphereLoader">
        <position>
          <X>0</X>
          <Y>0</Y>
          <Z>0</Z>
        </position>
        <rotation>
          <X>0</X>
          <Y>0</Y>
          <Z>0</Z>
        </rotation>
        ...
        <diameter>1</diameter>
      </SceneObjectLoader>
    </MyVariable>
  </variables>
  ...

  <sceneObjectLoaders>
    <MyVariable/>
  </sceneObjectLoaders>
<DataLoader>
```

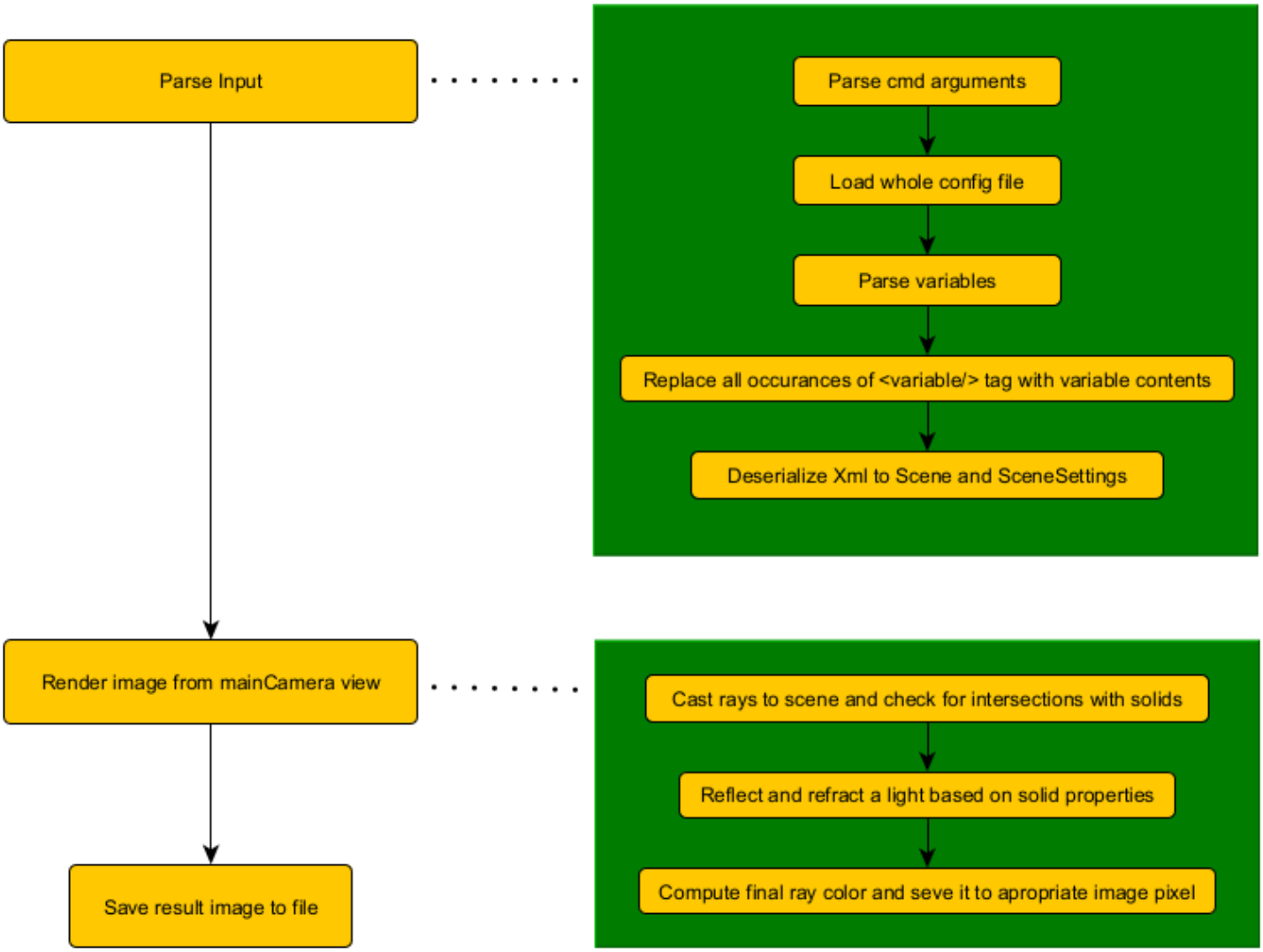
Programming Documentation

Program structure

The program is structured in a way, such that the program can be easily modified and added new SceneObjects and rendering types.



The figure describes the class structure of the program.



Flowchart of the program

The Flow of the program: On a Scene is called on mainCamera render function. The camera has specified LightModelComputation object which is used for rendering of each pixel separately. Objects, in scene are saved in a tree structure. Where inner tree nodes are represented by InnerSceneObjects and leaves are Solids.

Expandable classes

The program is prepared for expansion using inheritance of Camera, LightSource, Solid, Texture, Material, LightModelComputation. To add new objects with different functionality.

Camera

Camera specifies how and in what order are pixels rendered by Computation model set in RenderingSettings. The camera optionally the camera can be made to handle more advanced features for example multithreading or antialiasing.

LightSource

Represents object (usually invisible) emitting a light. The light can be emitted in different modes, for example: from point, from a plane parallel to each other.

Solid

Is a physical object interacting with light in a Scene.

Texture

Texture represents an information about a surface of a solid. Based on the position. For Image usage there needs to be an UV unwrap process.

Material

Defines surface properties of a Solid.

LightModelComputation

Computes light color and how does it reflect and refract on a solid surface.

Adding new class

When adding a new class, inheriting from a expandable class. There must be also created a loader class with non-parametric constructor and CreateInstance() method returning the added class. And the loader class must be registered to a loader of class, from which it is derived by tag [XmlInclude([TYPE OF THE NEW CLASS])] associated with the parent class.

```
[XmlInclude(typeof(ImageTexture)), XmlInclude(typeof(MonochromeTextureLoader))]
abstract public class TextureLoader{
```

```
  ...
}

public class ImageTextureLoader{
  public string path;

  ImageTexture CreateInstance(){
    return ImageTexture.FromFile(path);
  }
}
```

Nonexpendable classes

There is some classes, which are not intended to be inherited from. They are Scene, RendererSettings, ArgumentParser, Geometry, InnerSceneObject. But they can be directly modified.

InnerSceneObject

Represents an empty object used only for association purposes. To connect objects under one parent. which enables modifying their position and rotation in relation to the parent object.

Scene

Is a container for SceneObjects.

RendererSettings

Contains all settings for rendering with their default values.

Structures

There is couple of new structures usually wrappers to allow only geometrically sensible operations. And structures to ease calculations for common primitives.

Vector2D

Represents vector in 2 dimensional space. With overridden operators such that, they can interact only with appropriate types. Function Equals and operators == and != are overridden to direct comparison of values.

Add with	Result type
--as unary op--	Vector2D
Vector2D	Vector2D

Add with	Result type
Point2D	Point2D

Subtract with	Result type
--as unary op--	Vector2D
Vector2D	Vector2D
Point2D	Point2D

Product with	Result type
Vector2D	Vector2D
Vector2d	Vector2D
double	Vector2D

Division with	Result type
Vector2D	Vector2D
double	Vector2D

Vector3D

Represents vector in 3 dimensional space. With overridden operators such that, they can interact only with appropriate types. Function Equals and operators == and != are overridden to direct comparison of values.

Add with	Result type
Vector3D	Vector3D
Point3D	Point3D

Subtract with	Result type
--as unary op--	Vector3D

Subtract with	Result type
Vector3D	Vector3D
Point3D	Point3D

Product with	Result type
Vector3D	Vector3D
Vector3d	Vector3D
double	Vector3D

Division with	Result type
Vector3D	Vector3D
double	Vector3D

Point2D

Represents point in 2 dimensional space with double precision. It wraps Vector2d and overrides operators such that, they can interact only with only appropriate types. Function Equals and operators ==, != are overridden to use direct comparison of values not pointer comparison.

Note: Vector2d is imported structure from OpenTK.Mathematics

Add with	Result type
Vector2D	Point2D

Subtract with	Result type
Point2D	Vector2D
Vector2D	Point2D

Product with	Result type
Vector2d	Vector2D

Product with	Result type
double	Point2D

Division with	Result type
Vector2d	Point2D
double	Point2D

Point3D

Represents point in 3 dimensional space with double precision. It wraps Vector3d and overrides operators such that, they can interact only with only appropriate types. Function Equals and operators ==, != are overridden to use direct comparison of values not pointer comparison.

Note: Vector2d is imported structure from OpenTK.Mathematics

Add with	Result type
Vector2D	Point2D

Subtract with	Result type
Point2D	Vector2D
Vector2D	Point2D

Product with	Result type
Vector2d	Vector2D
double	Point2D

Division with	Result type
Vector2d	Point2D
double	Point2D

Ray

Ray represents a half-line by point and a vector. It is used in scene ray casting to calculate intersections with objects.

BasicPlane

BasicPlane represents infinite plane by a point on the plane and normal vector. It can find interception between two planes of a plane and a ray.

Specification

The program should be ray-tracing renderer for basic geometrical 3D shapes. Capable of computing antialiasing, reflections, refractions and being easely expandable.

The program will have "output" and "config" command line arguments, which sets path to files. The output file can be specified also in a config file, but the cmd argument is prefered.

Command Line Argments

The program will receve path to XML configuration file, containing definition of a scene, and path to output image file of HDR type .pfm. It will be created if doesn't exists.

Rendering

The rendered image is created by camera object in a scene in a prespctive view. During ray-traceing the reflection and the refraction rays are considered and traced. While the ray color contribution is noticible or the recursion is not too deep. And The maximal depth is configurable. The defined objecs are spheres and planes, but it should be easy to add other objects.

Scene definition

The scene is defined in config file using XML.

Output

The ouput is an image at path specified by cmd arumment or config file of HDR type .pfm.

Perforamce

The performance is dependent on applied setting. Mainly on the camera resolution and max ray recursion depth.

Libraries

The program uses only standard library. Notable is System.Xml.Serialization for parsing XML config file.