Project documentation

Path tracer 3

Overview

In this project, we planned and implemented a path tracer. Our main goal was that our path tracer could render images of different environments that include objects of any shape. We also wanted our path tracer to be user friendly so one of our goals was to make the user interface simple to use.

To be able to have objects of any shape in our scene environments we opted to use triangle mesh to construct our objects. The triangles support assigning normal vectors for each of the vertices separately; this allows us to use Phong normal interpolation. This approach produces smooth curved surfaces even with a relatively low number of triangles.

The software supports 3 different material types: diffuse, specular and transparent. The transparent material is not physical due to unresolved problems with refraction. Each of the material options has its own properties that can be fine tuned to make each material have different variations. For example, the width of the distribution in the bidirectional reflectance distribution function (BRDF) can be altered for diffuse surfaces. Any mesh object can be declared to be a light source by giving it appropriate material parameters.

Our program features a command line interface. The main function takes a filename pointing to a scene file as an argument. These files store the information about the camera, materials and objects in an environment. The objects' triangle meshes are stored in separate files.

Our path tracer parses the information of camera location and direction in the rendered environment from the scene file. It also parses the location and material of each object in the rendered environment and location of triangle mesh information from a scene file. Then the triangle mesh is parsed separately for each object from their own files.

The program outputs the rendered image into a png-file.

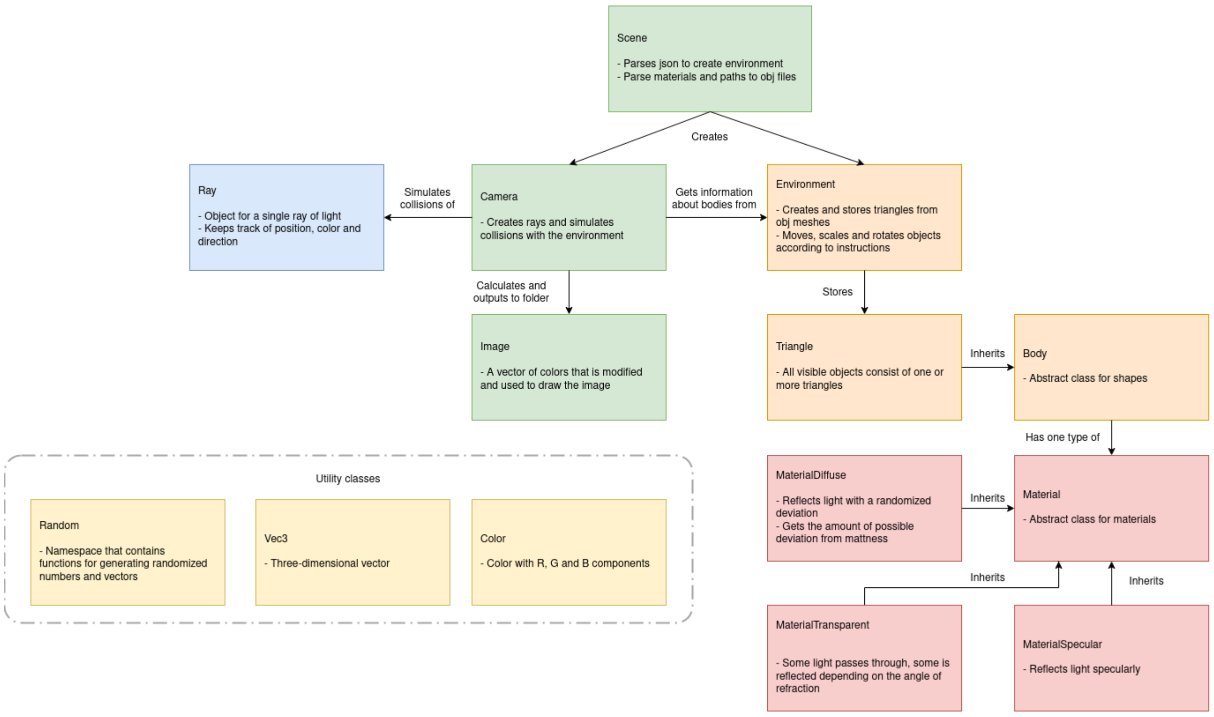
Our path tracer uses Monte Carlo integration to allow modeling scattering from diffuse and transparent surfaces. The number of samples per pixel can be given as a command line argument.

We use OpenMP to parallelize rendering on all CPU cores available to make rendering fast. We do not use any other sort of explicit parallelization; for example no vectorization is used apart from what the compiler might do when given an optimization flag. Also, we do not use any method for subdividing the 3D space - such as an octree - to obtain performance benefits; this is something that could be optimized further if we were to continue working on the project.

Software structure

Class interfaces are documented in the Attachments under Class Interfaces-heading.

Class relationship diagram



Scene
- Parses json to create environment
- Parse materials and paths to obj files
Creates
Ray
- Object for a single ray of light
- Keeps track of position, color and direction
Simulates collisions of
Camera
- Creates rays and simulates collisions with the environment
Gets information about bodies from
Environment
- Creates and stores triangles from obj meshes
- Moves, scales and rotates objects according to instructions
Calculates and outputs to folder
Stores
Image
- A vector of colors that is modified and used to draw the image
Triangle
- All visible objects consist of one or more triangles
Inherits
Body
- Abstract class for shapes
Has one type of
Utility classes
Random
- Namespace that contains functions for generating randomized numbers and vectors
Vec3
- Three-dimensional vector
Color
- Color with R, G and B components
MaterialDiffuse
- Reflects light with a randomized deviation
- Gets the amount of possible deviation from mattness
Inherits
Material
- Abstract class for materials
Inherits
Inherits
MaterialTransparent
- Some light passes through, some is reflected depending on the angle of refraction
MaterialSpecular
- Reflects light specularly

External libraries

During our project planning we saw it best to start from the scratch and write as much of the path tracer by ourselves as we could. This way we could use more time on learning and creating rather than reading different library documentations. However not everything could be written by ourselves in the timeframe we had, so there are few small libraries we decided to use to help with certain tasks in addition to the standard library.

Image output

We first started out with .ppm images but later decided .png images would be more universal and more efficient. For this we used a single header file library from a larger collection of single file header libraries called stb.

https://github.com/nothings/stb

From the collection we used stb_image_write -library which is an image writing library that supports a few universal image formats. The library has its own easy to use interface. The interface takes filename, image width, image height, number of colour channels, array of colours and width of each row in bits for its parameters for png images. Using this information, it writes an image to the disk.

JSON

To store the information of our rendered environment we opted to use JSON files. JSON is a very readable form for storing data, which makes it also easy to modify. Downside of this is that it makes it a bit harder to parse into easily usable data as there can be some variability. Because storing data is not really the core part of path tracing, we opted to rather use a well written and tested single header file JSON library rather than write a moderate one by ourselves. We used a library called JSON.

https://github.com/nlohmann/json

The library provided an easy to use interface. The library parses the information from the JSON file to a JSON class defined by the library. A parsed JSON file can be easily accessed by the variable names defined in the JSON file.

Instructions for building and using the software

Compilation

The project uses a Makefile for compilation. One can compile the program using one of the following commands:

make

- A “release”-version of the program, optimized using compiler flag -O3.

make debug

- A debug version that can be used with GDB for example (compiled with the flag -g)

make sanitize

- A debug version which utilizes AddressSanitizer (https://github.com/google/sanitizers/wiki/AddressSanitizer) to check for memory errors. (One must have AddressSanitizer installed to compile and run the executable)

Dependencies

One must have **make** and **gcc** installed to compile the program.

The program uses **OpenMP** (https://www.openmp.org/) for CPU multithreading, so that must be installed as well for the program to compile and work.

We have developed the program using Ubuntu 20.04. We haven't tested the code on other platforms, but we guess that it should be fairly portable at least across different Linux distributions.

User guide

Basic usage after compiling the program is as follows:

./pathtracer [scenefile] [samples per pixel]

where

- [scenefile] is a path to a JSON-file containing the geometry of the environment.
- [samples per pixel] is the number of rays used per pixel. This is an optional argument with a default value of 1. Higher number of rays produces images with less noise when diffuse or transparent surfaces are used.

The program writes the rendered image to a png file. The files are named depending on the system time, using format YY_MM_DD-hh_mm_ss.png.

Using JSON-scenefiles

The basic structure and fields for the scene files are as follows:

- *camera*
  - *focalPoint*: The focal point of the camera; in other words, the origin of the rays.
  - *direction*: A vector with fields x, y and z. Describes in which direction the camera is pointed at.
  - *xReso, yReso*: Number of pixel columns and rows (respectively) in the image that is produced.
- *materials*
  - *comment*: A free-form text field containing some description for the material.
  - *type*: Either “specular”, “diffuse” or “transparent”.
  - *mattness*: A parameter for materials of type “diffuse”. Specifies the width of the distribution of the material's BRDF. Must be between 0 and 1; 0 behaves like specular, 1 like completely diffuse material.
  - *refIndex*: A parameter for materials of type “transparent”. Specifies the refractive index of the material.
  - *color*: A field with 3 values, r, g and b, each between 0 and 1. Specifies the color of the material.
  - *isLuminous*: “true” if the material is a light source, “false” otherwise.
- *objects*
  - *comment*: A free-form text field containing some description for the object.
  - *type*: Type of the object; only accepts “mesh” for now.
  - *path*: A relative file path pointing to the .obj file that the mesh is loaded from.

- *materialIndex*: Specifies the material. For example: if this is 2, the mesh uses the 3rd material defined in the same file.
- *midPoint*: Center of the bounding box in the scene.
- *height*: Height of the object in the scene.
- *xrot, yrot, zrot*: Allow rotating the original .obj-file around the coordinate axes by a specified amount of degrees

Examples of the scenefiles can be found in the /src/environments -folder. The user is encouraged to experiment by modifying the parameters given in the files.

Testing

The memory management of the program was tested with Valgrind by running the "make debug" compiled executable with --leak_check=full.

In addition to the demo images, each material was tested with a designated test image. The test image consists of ten triangles with a different angle in [0, 90] degrees. Images are included in the Attachments section.

For the specular material the tests were really simple. The stack of triangles was illuminated with a wide (10000 units, the height of the stack was 10 units) wall of light. In the first test the wall was behind the camera, in second on the right side of the stack. As assumed, in the first picture half of the triangles were completely illuminated. In the second test all except half of the transverse top triangle were illuminated.

For the diffuse material tests were done as in the first case with the specular material. Six tests were done with mattness in [0, 1]. Mattness worked also as intended, brightness of the material decreasing as a function of both mattness and angle. Mattness 0 worked as a specular material as expected.

For the transparent material the direction of the normal of the triangle is a factor to consider, as it defines the inside and outside of the material. When the rays are incident on the surface from the "outside", they should be reflected by the surface with some probability p that depends on the angle of incidence, and transmitted with probability 1-p. Coming from the inside there should be one more possiblity, i.e. total reflection on large enough angles of incidence. The tests were conducted with a similar setup as the previous two. In these tests the transparent material worked as expected, with the surface reflecting back a small portion of the light. The total reflection can also be seen in images 10, 12, 14. However, on a more complicated test image 15 there is a straight glass panel over the demo image 2. As can be seen, the panel magnifies the scene behind it, which shouldn't happen with a straight panel.

Work log

Time tracking

	Sprint 1			Sprint 2			Sprint 3			Project
	44	45	Total	46	47	Total	48	49	Total	Total
Anton Pirhonen	10	12	22	12	10	22	13	10.5	23.5	67.5
Atso Ikäheimo	10	14	24	8	14	22	10	12	22	66
Joona Munukka	12	9	21	8	10	18	14	10	24	63
Kalle Jyrkinen	10	8	18	6	12	18	18	10	28	64
Kerttu Aronen	8	14	22	10	11	21	12	9	21	64

Overview on the division of work

We managed our project in a Scrum-like fashion. The period of six weeks was divided into three sprints each lasting two weeks. We arranged weekly meetings on Tuesdays to collaborate and discuss the work done, possible problems and planned work.

In the beginning of each week we created tasks or backlog items related to the objectives we wanted to accomplish in the project plan. We used Trello for backlog management. Tasks were also added to the backlog outside of the original plan, as we discovered more about what we wanted to accomplish during the course. In the beginning of a sprint each team member was given the responsibility over a certain task(s).

Part of the work that is not visible in the backlog is the reviewing of Pull Requests. There were also many minor tasks that were completed while working on the major backlog items that are not added to the backlog on their own.

Personal Descriptions

Anton Pirhonen

Week 44 - I studied what it requires to build a path tracer. I worked on the project plan to design the planned objectives and the initial class structure. I also studied the Wavefront .obj-format to prepare for reading triangle meshes.

Week 45 - I created an initial Parser and methods to add Triangles to the Environment class.

Week 46 - I studied Phong-interpolation for interpolating the normal vector from vertex normals to allow curved surfaces. I worked on updating the parser to add the normal vectors and fixed issues related to it. I worked on implementing the Phong Interpolation

Week 47 - I finished the Phong interpolation.

Week 48 - I collaborated with Joona to design the initial scene input JSON-format. We pair-coded the parser that creates the objects to the environment from the JSON-file.

Week 49 - I worked on documenting the project, creating demo images with our path tracer. I added error/exception handling to the parser.

Kalle Jyrkinen

Week 44 - Studying the algorithm and the mathematics behind it. Creating basic project structure and a simple makefile. Writing methods for Vec3.

Week 45 - More mathematics and studying methods for image rendering (Phong interpolation for example). Writing methods for Sphere (not included in the final version - we swapped to using only triangles after this week)

Week 46 - Writing algorithms for random vector generation that Joona implemented later. More studying about different methods for shading.

Week 47 - Monte Carlo integration. Refactoring color production and Camera.

Week 48 - CPU parallelization. Ability to move, scale and rotate objects in a scene. Major refactoring here and there & several bug fixes. Creating scenes with custom objects.

Week 49 - Writing project documentation. Writing a new makefile. Creating an ASCII-art progress bar.

Joona Munukka

Week 44 – I read about the basics of path tracing and created a base for body-class and its subclasses.

Week 45 – I studied different ways to use image libraries to output images in different standard formats. I also started looking into random number generations and created a base for different random number and vector functions.

Week 46 – I finished random functions with the help of Kalle.

Week 47 – I implemented a feature that outputs our rendered scene in PNG-format with the help of image library. I also read about manipulating objects in space and got more familiar with the structure of our path tracer.

Week 48 – I worked together with Anton to build the system we use to handle loading different scenes with JSON files.

Week 49 - I worked on documenting the project and I created demo scenes and images with our path tracer.

Atso Ikäheimo

Week 44 - I studied the basics of path tracing and wavefront .obj formats. Drew class diagram. Implemented the first version of Camera class.

Week 45 - I implemented the first version of a Triangle class and collision detection. Continued work on CameraSimple with Kerttu and drew first pictures with the help of Anton.

Week 46 - I implemented basic materials: specular and diffuse with Kerttu.

Week 47 - I implemented the first version of a transparent material with Kerttu.

Week 48 - I continued work on transparent material with Kerttu. Upgraded CameraSimple to Camera with qol-improvements.

Week 49 - I continued work on transparent material with Kerttu. Created demo and test scenes and wrote documentation. Cleaned up code.

Kerttu Aronen

Week 44 - Studied the basics of path tracing. Implemented the first version of Ray class.

Week 45 - Continued work on CameraSimple with Atso. Looked into the physics of materials.

Week 46 - Implemented the specular and diffuse materials with Atso. Started to work on transparent material.

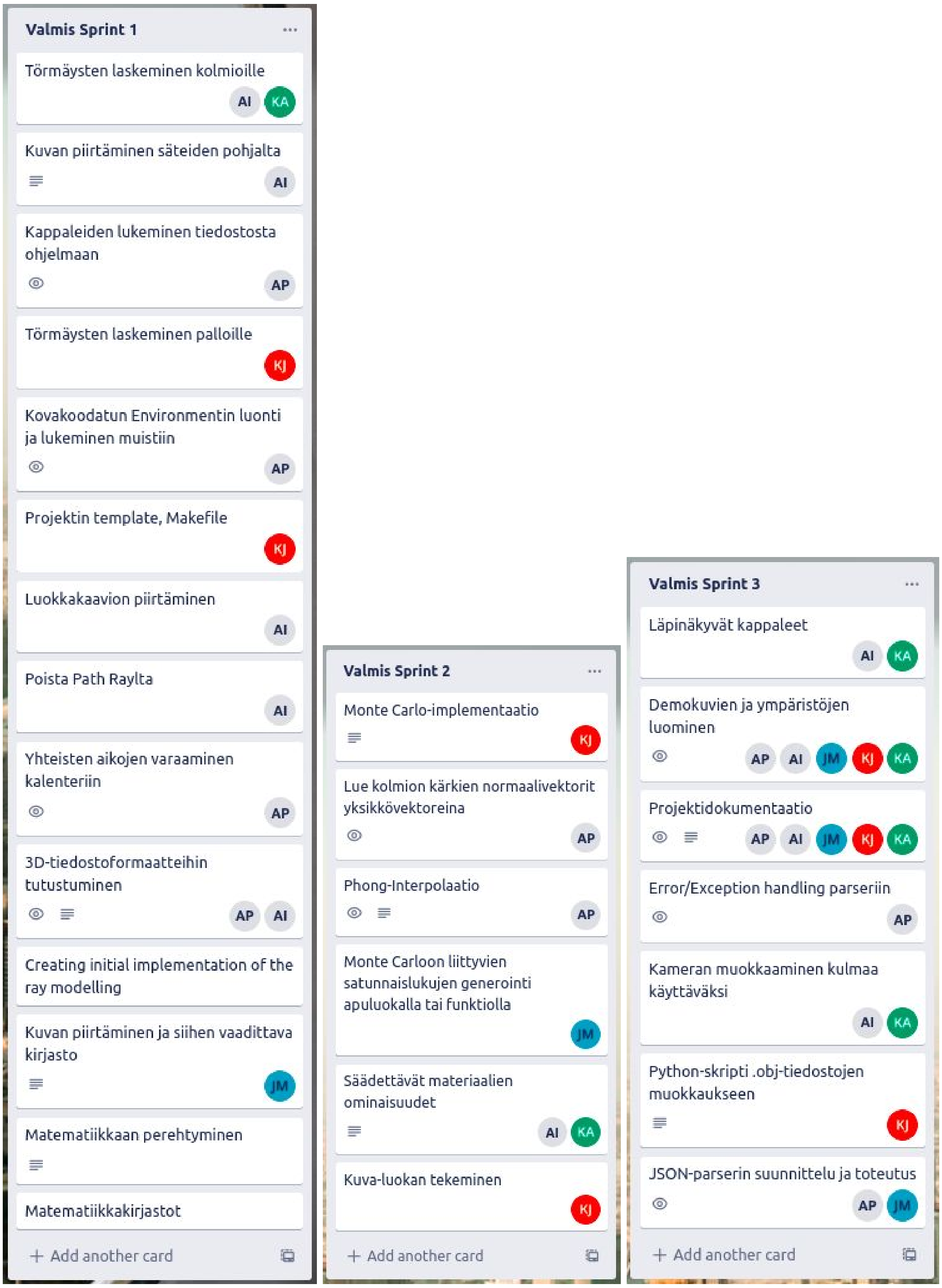
Week 47 - I implemented the first version of a transparent material with Atso.

Week 48 - Continued work on transparent material with Atso. Worked on the implementation of Camera from CameraSimple.

Week 49 - Continued to debug transparent material with Atso. Class documentation and testing documentation.

Due to technical reasons I did a major part of my work on Atso’s computer and git, so most of my commits are pushed on his git.

Trello board used for project and backlog management



Valmis Sprint 1

- Törmäysten laskeminen kolmioille — AI KA
- Kuvan piirtäminen säteiden pohjalta — AI
- Kappaleiden lukeminen tiedostosta ohjelmaan — AP
- Törmäysten laskeminen palloille — KJ
- Kovakoodatun Environmentin luonti ja lukeminen muistiin — AP
- Projektin template, Makefile — KJ
- Luokkakaavion piirtäminen — AI
- Poista Path Raylta — AI
- Yhteisten aikojen varaaminen kalenteriin — AP
- 3D-tiedostoformaatteihin tutustuminen — AP AI
- Creating initial implementation of the ray modelling
- Kuvan piirtäminen ja siihen vaadittava kirjasto — JM
- Matematiikkaan perehtyminen
- Matematiikkakirjastot

+ Add another card

Valmis Sprint 2

- Monte Carlo-implementaatio — KJ
- Lue kolmion kärkien normaalivektorit yksikkövektoreina — AP
- Phong-Interpolaatio — AP
- Monte Carloon liittyvien satunnaislukujen generointi apuluokalla tai funktiolla — JM
- Säädettävät materiaalien ominaisuudet — AI KA
- Kuva-luokan tekeminen — KJ

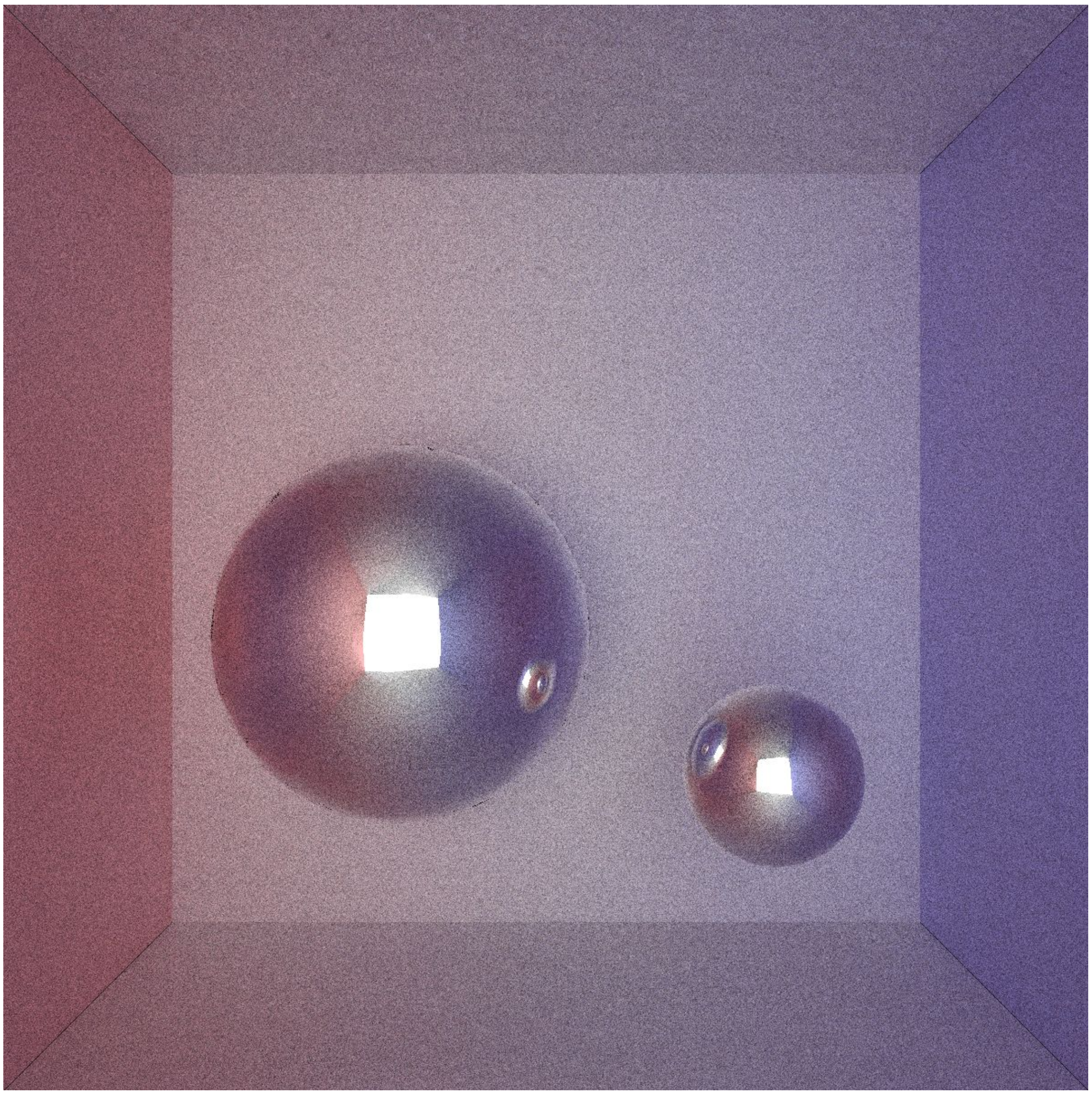
+ Add another card

Valmis Sprint 3

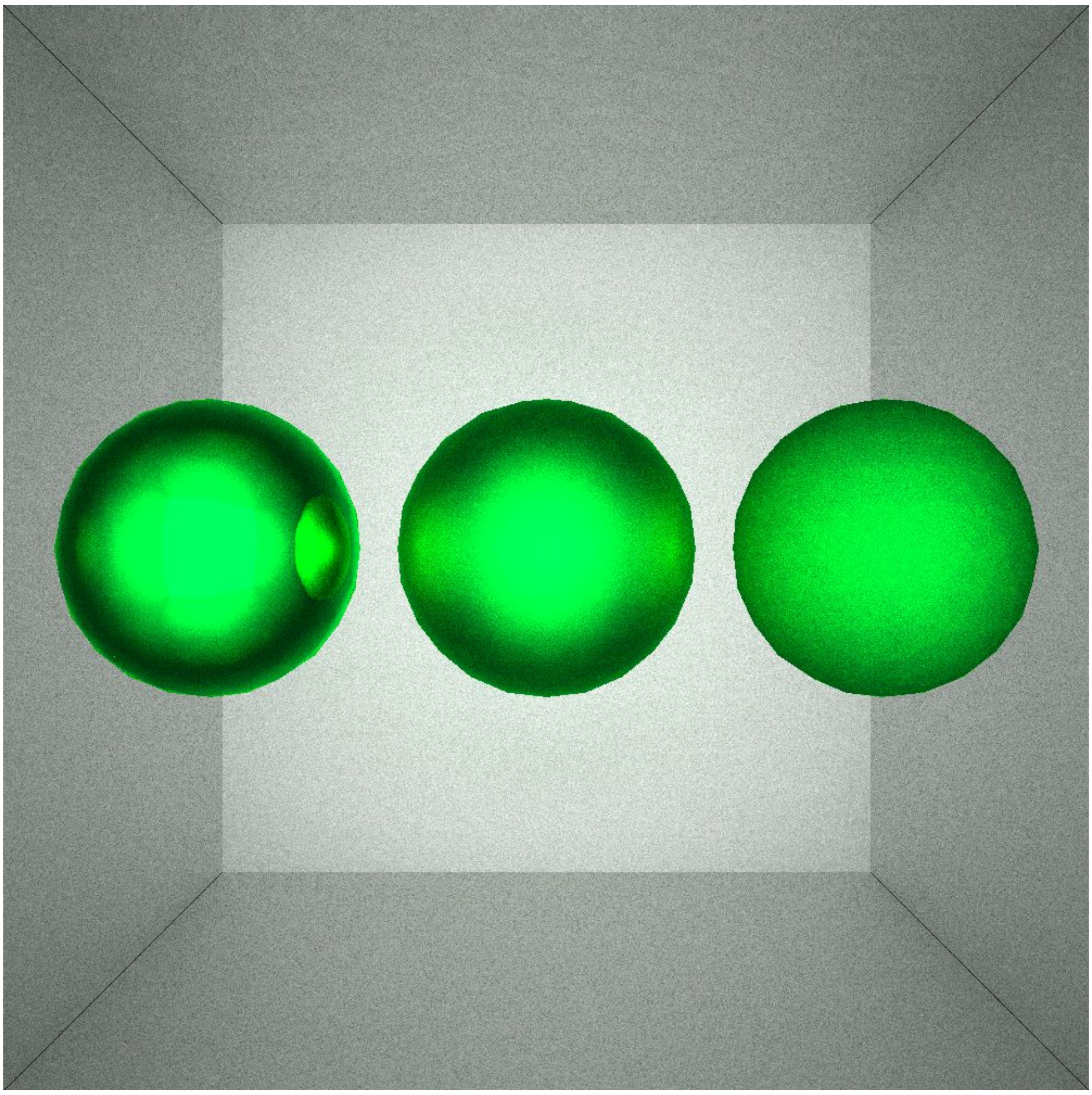
- Läpinäkyvät kappaleet — AI KA
- Demokuvien ja ympäristöjen luominen — AP AI JM KJ KA
- Projektidokumentaatio — AP AI JM KJ KA
- Error/Exception handling parseriin — AP
- Kameran muokkaaminen kulmaa käyttäväksi — AI KA
- Python-skripti .obj-tiedostojen muokkaukseen — KJ
- JSON-parserin suunnittelu ja toteutus — AP JM

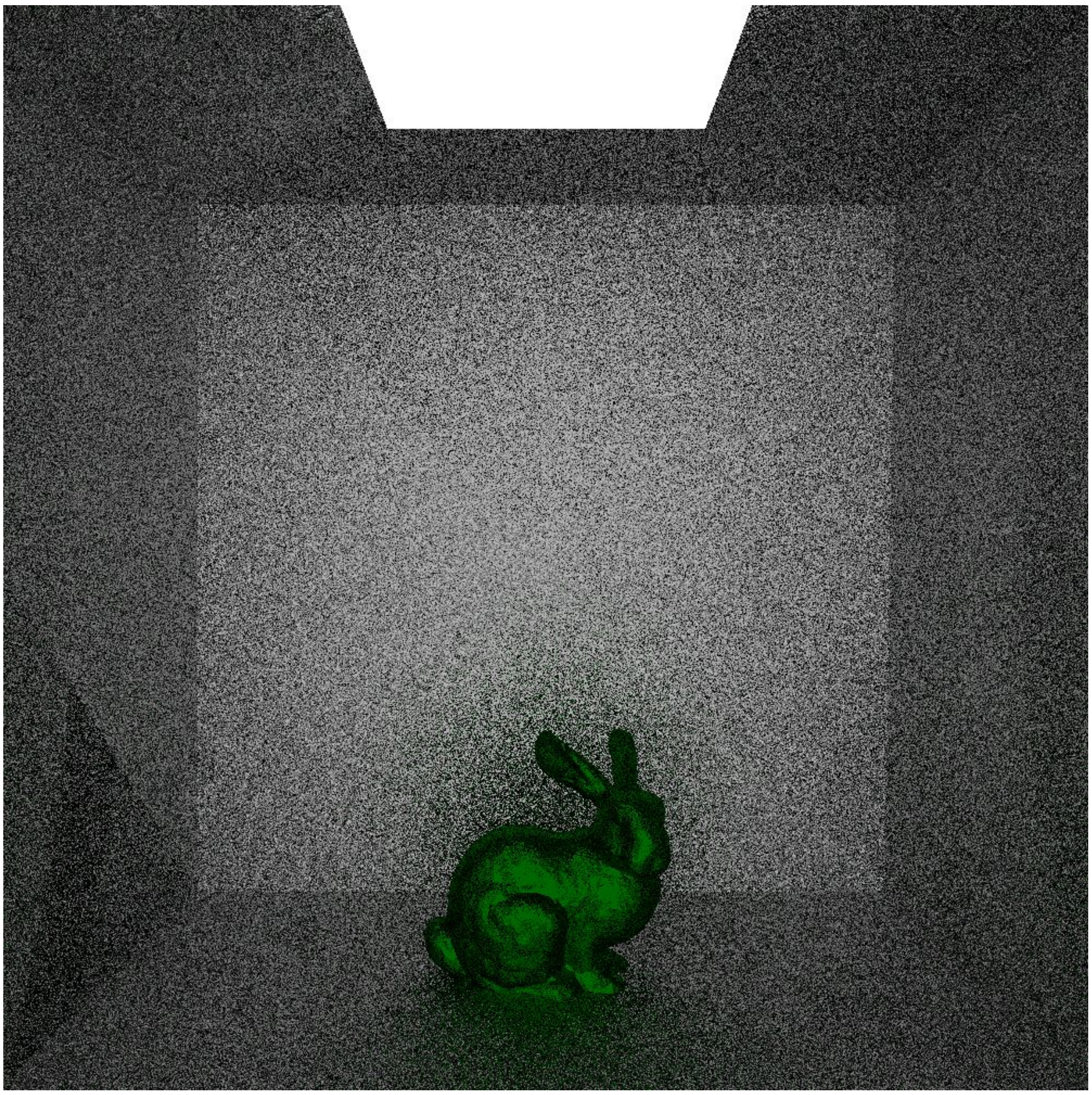
+ Add another card

Attachments

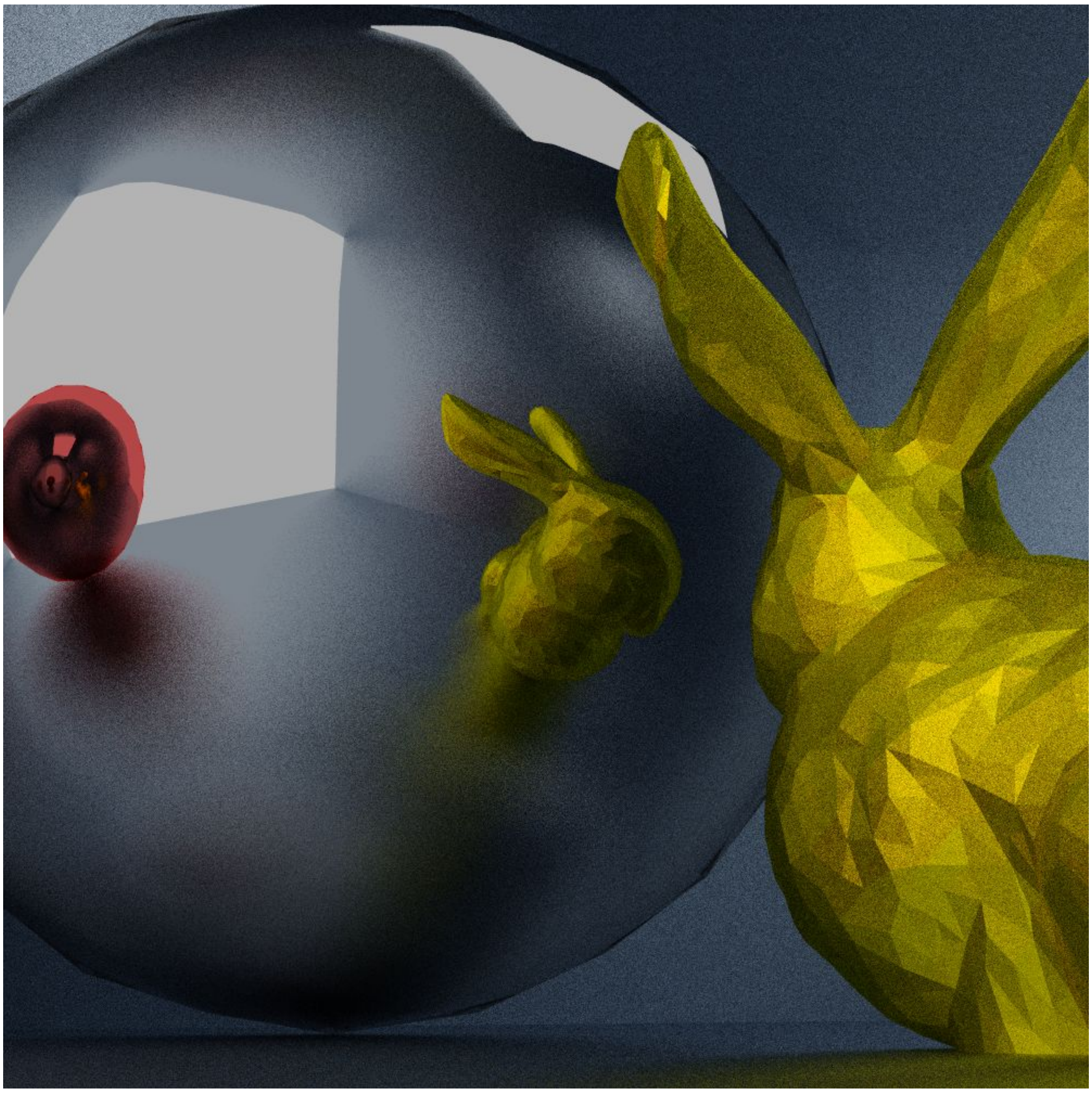


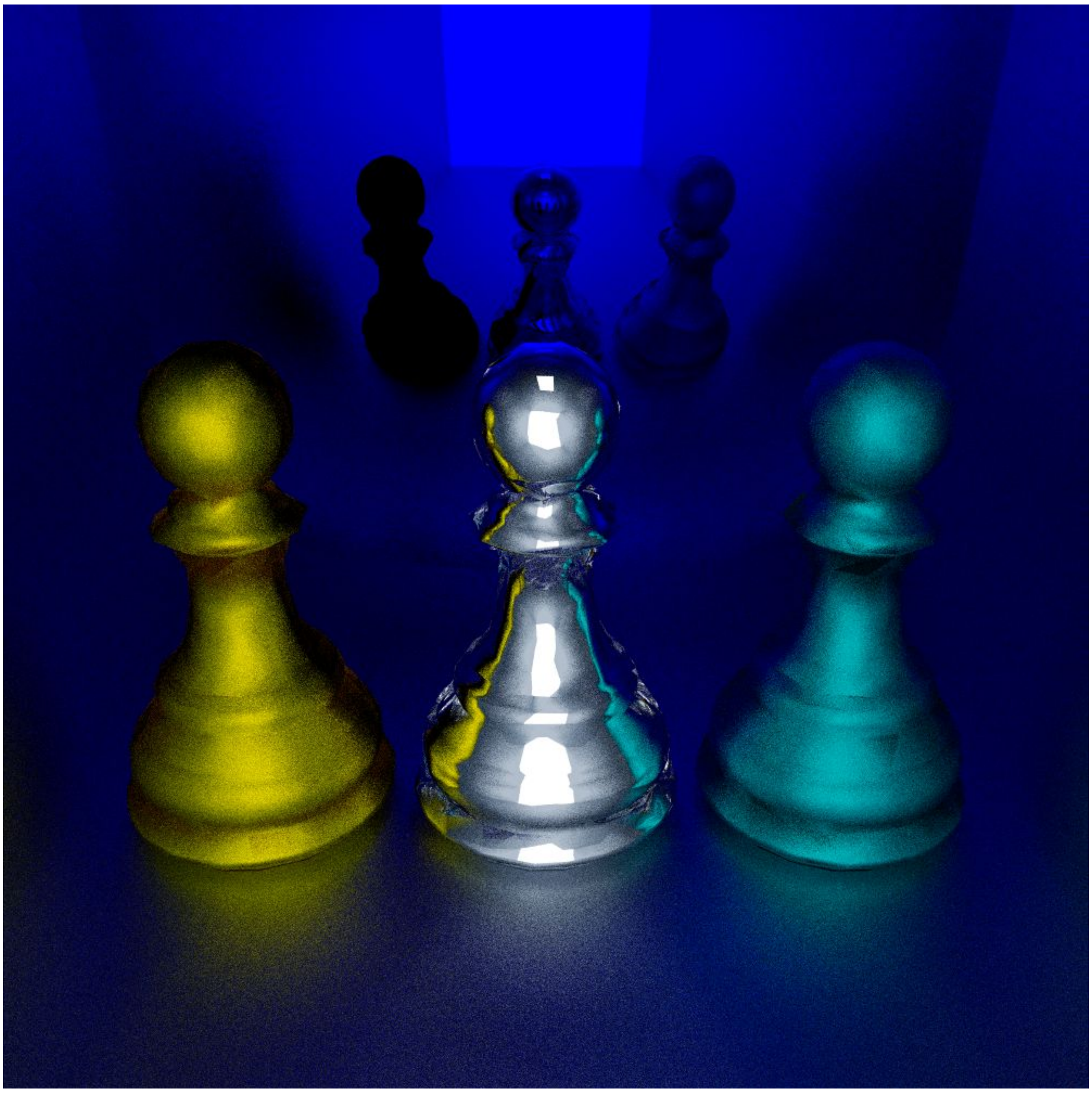
Demo 1. Two specular balls in a diffuse box


Demo 2. Three green balls with different mattnesses in a diffuse box.



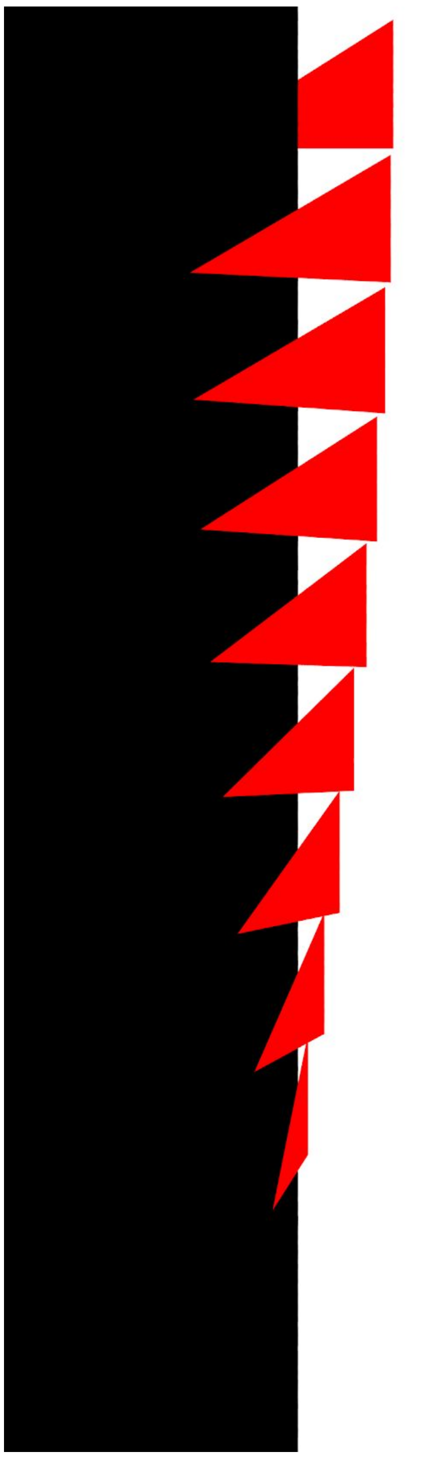
Demo 3. Diffuse green bunny in a diffuse box with two light sources

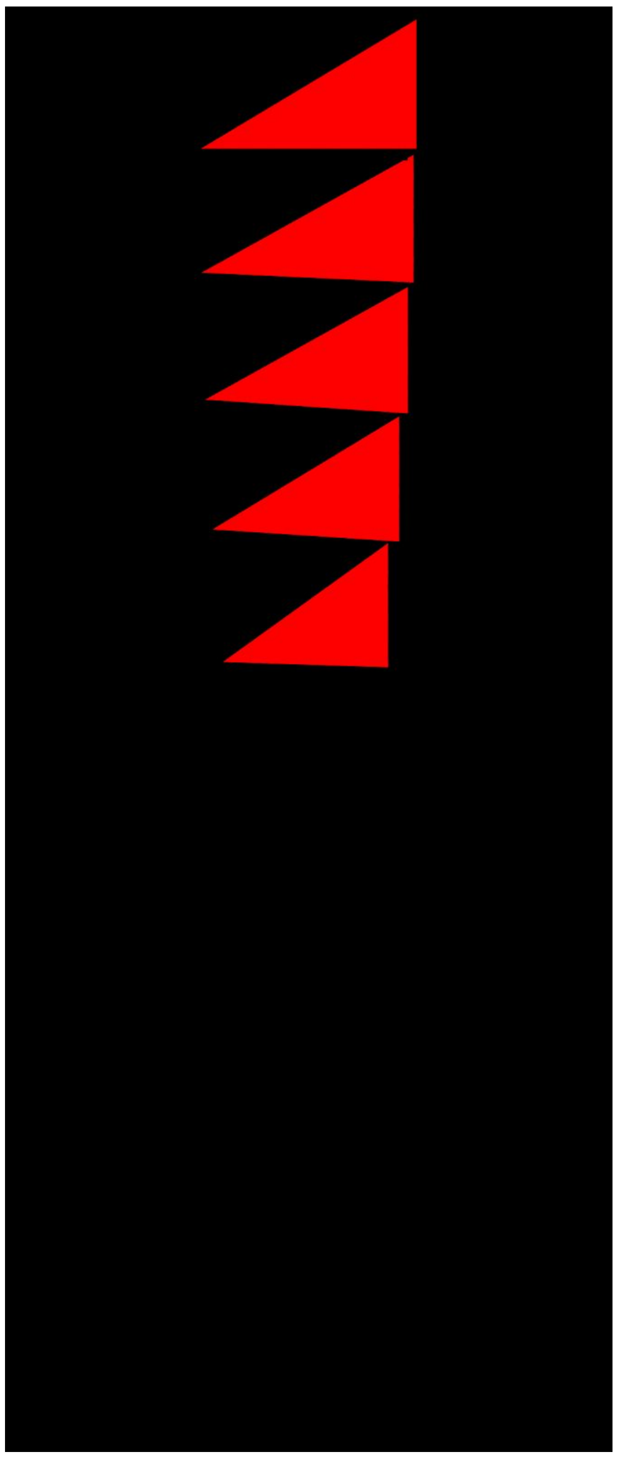

Demo 4: Two specular balls and a bunny in a bluish diffuse box with two light sources. The fact that the objects consist from triangles becomes more evident as the number of samples of the Monte Carlo integration is increased. To avoid this one would have to use objects with more triangles.

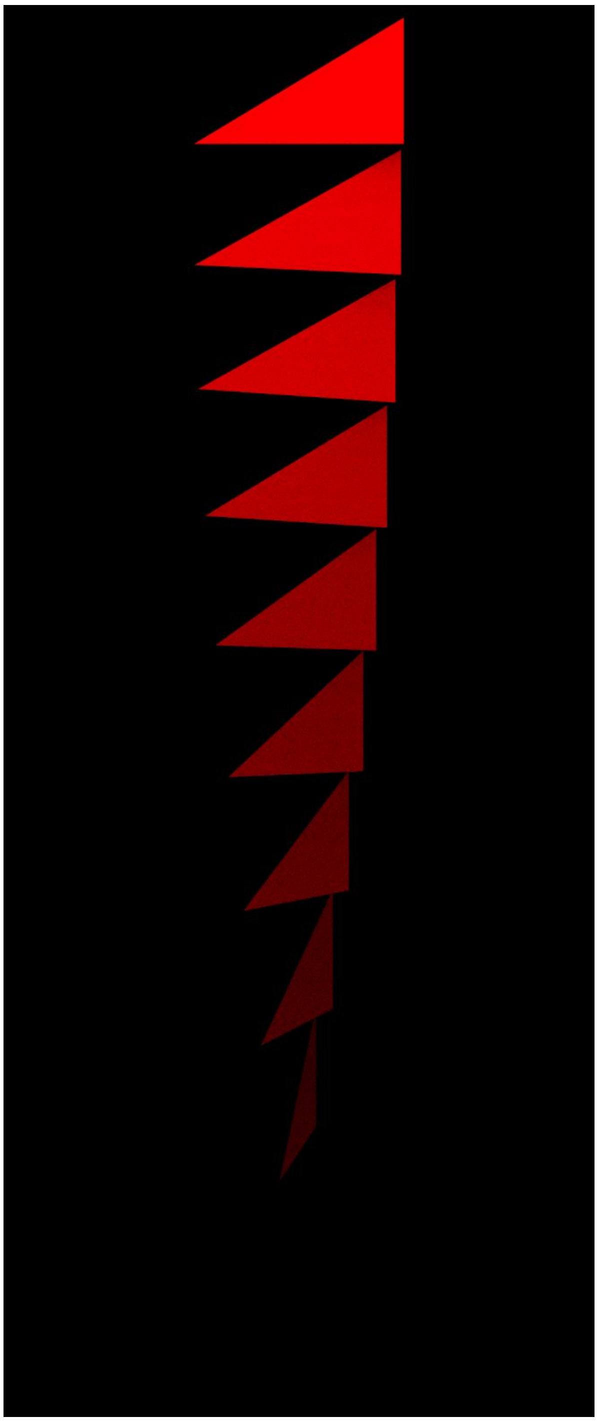


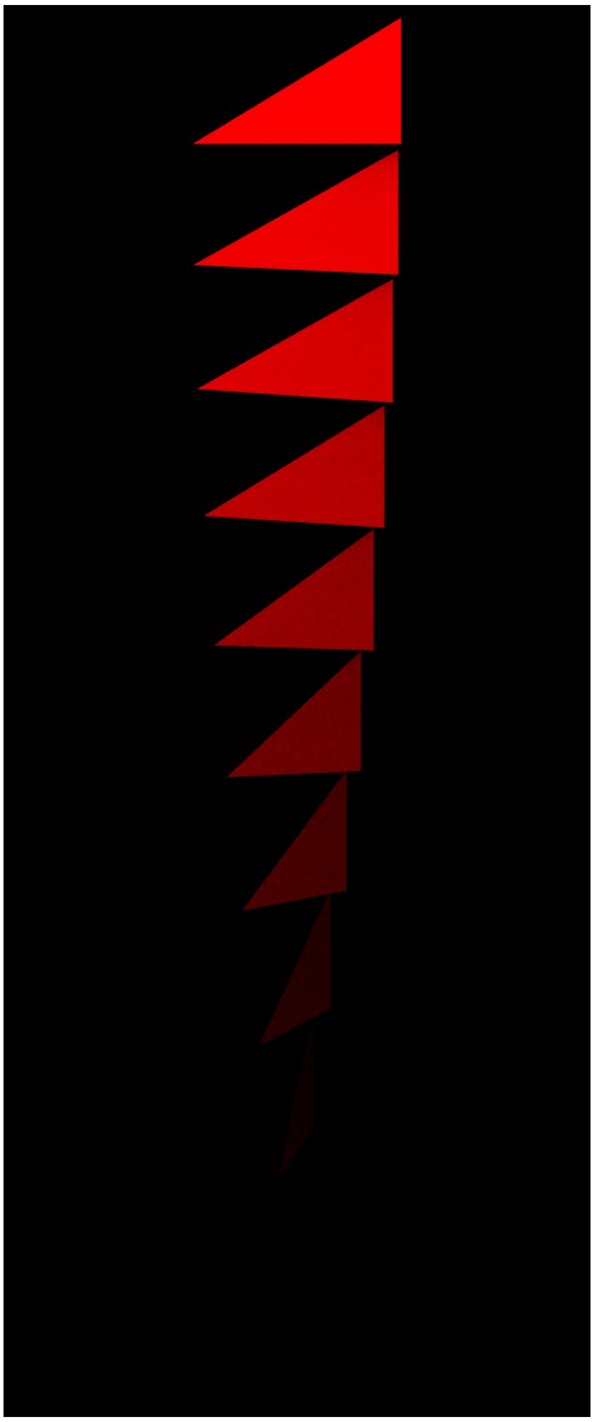
Demo 5: Three pawns in a diffuse white box. There are two light sources similar to demos 3 and 4, and the back wall is blue, convex and specular. Please note a property of RGB colors: yellow does not reflect at all from a blue surface.

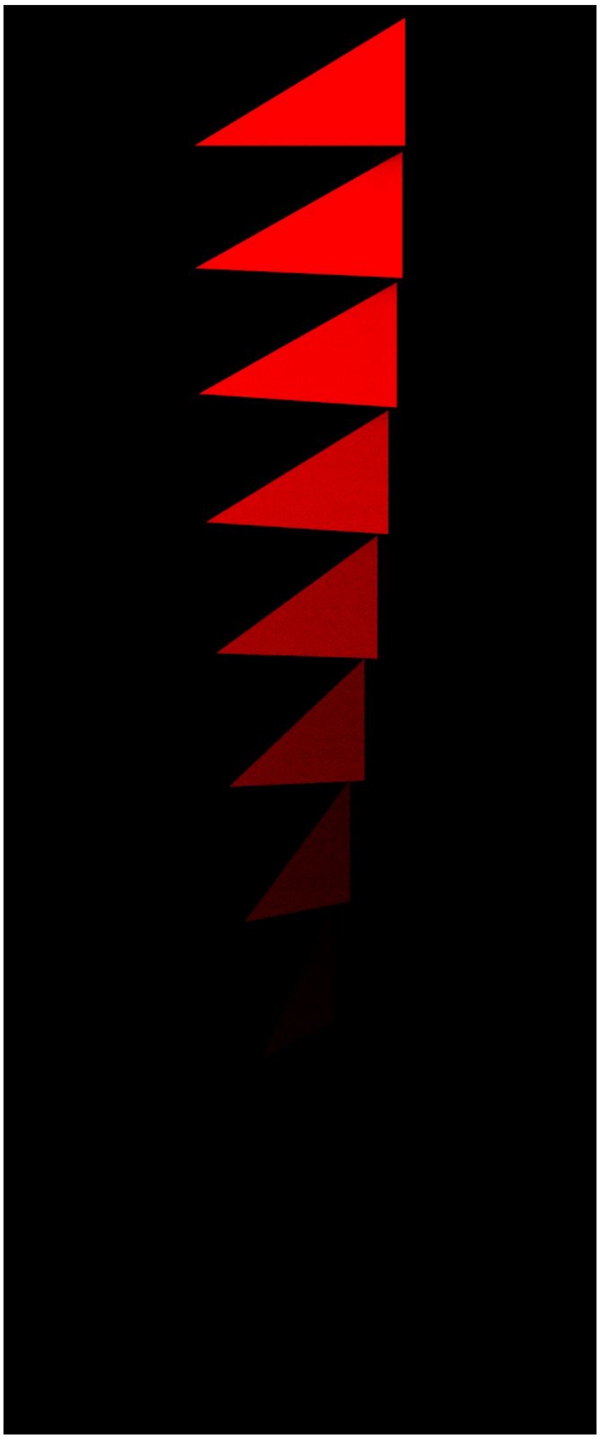
Test images

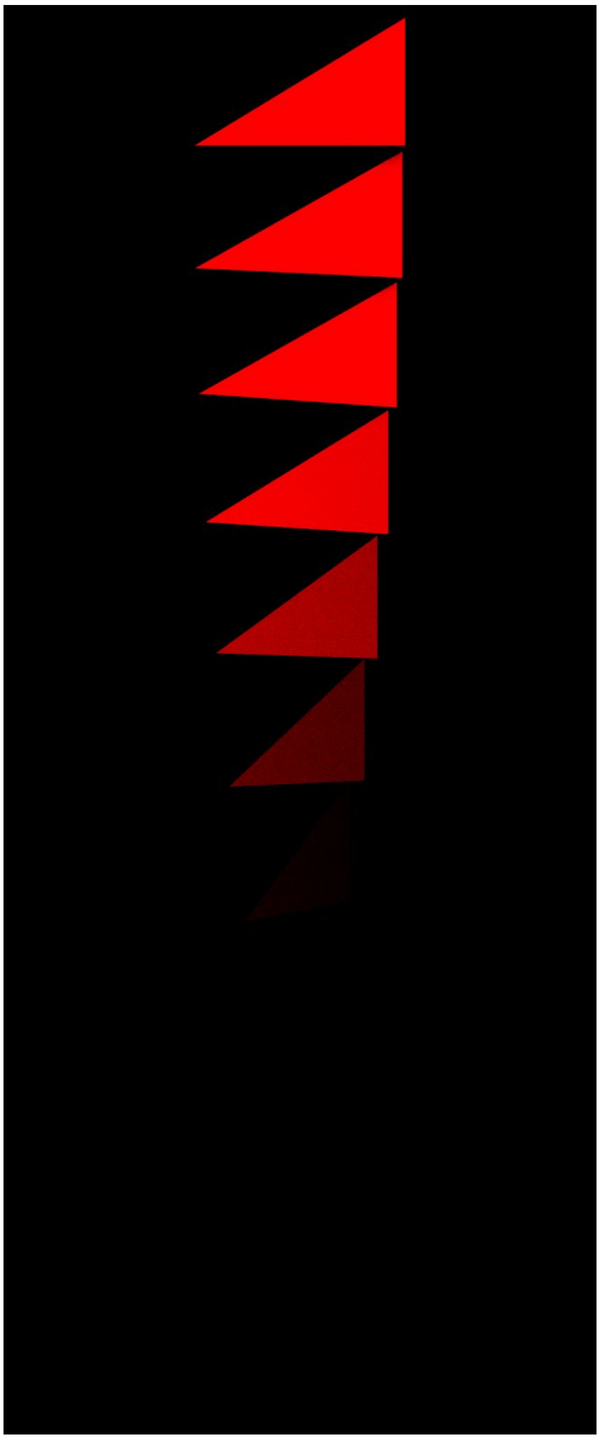

1. Specular material, light on side

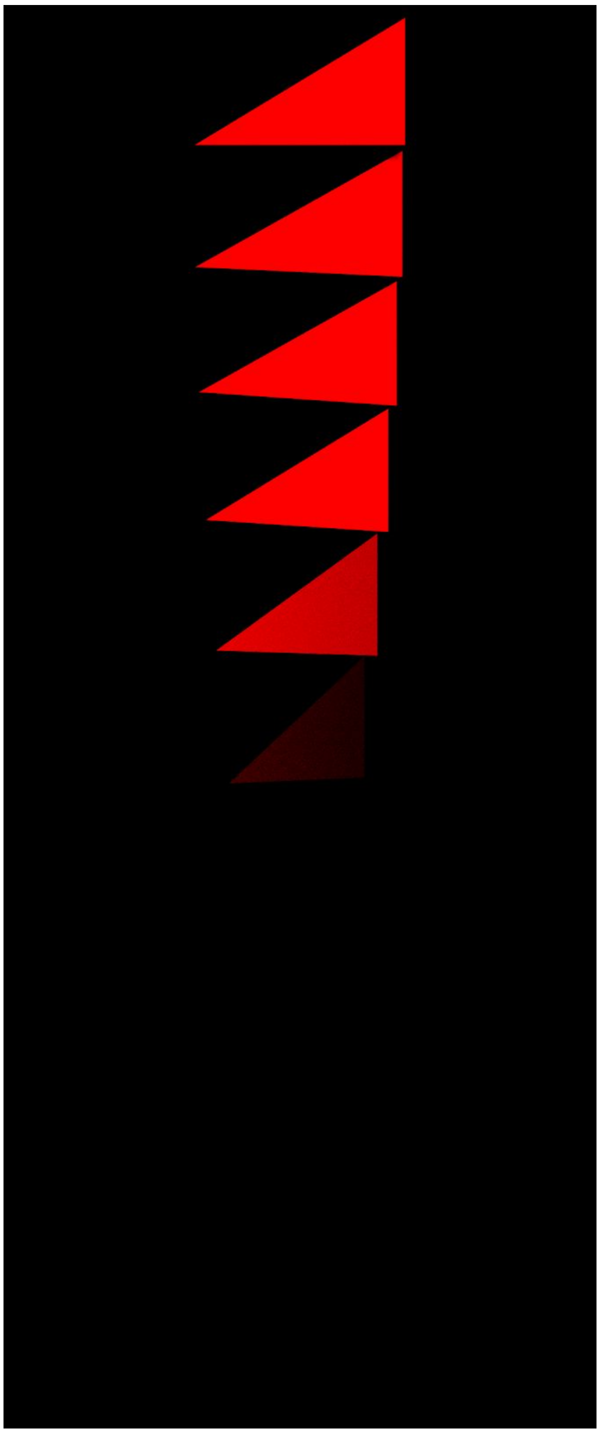

2. Specular material, light behind the camera

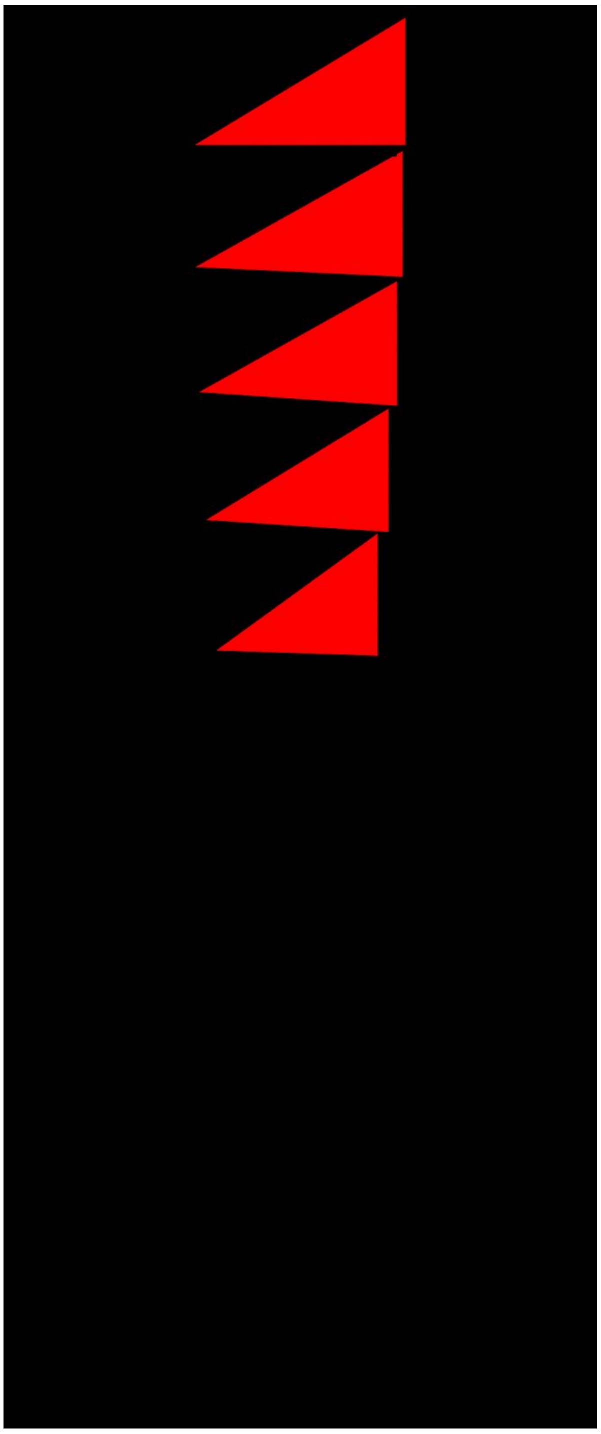

3. Diffuse material, mattness = 1,

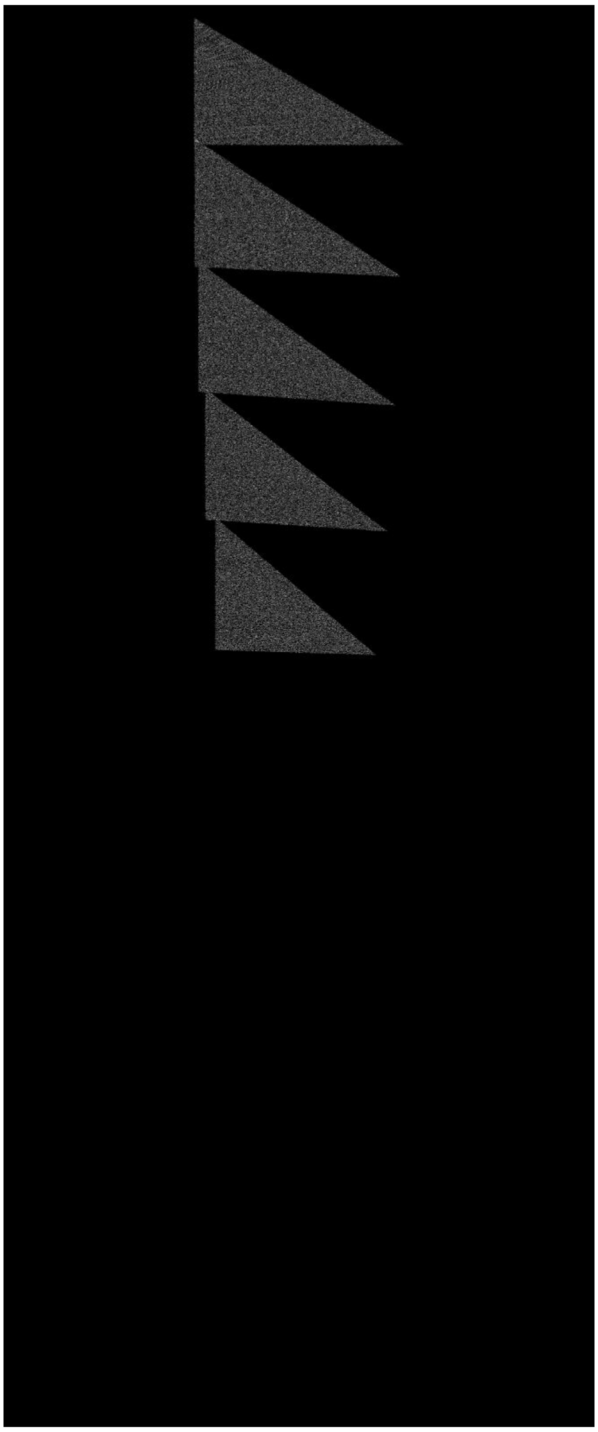

4. Diffuse material, mattness = 0.8


5. Diffuse material, mattness = 0.6

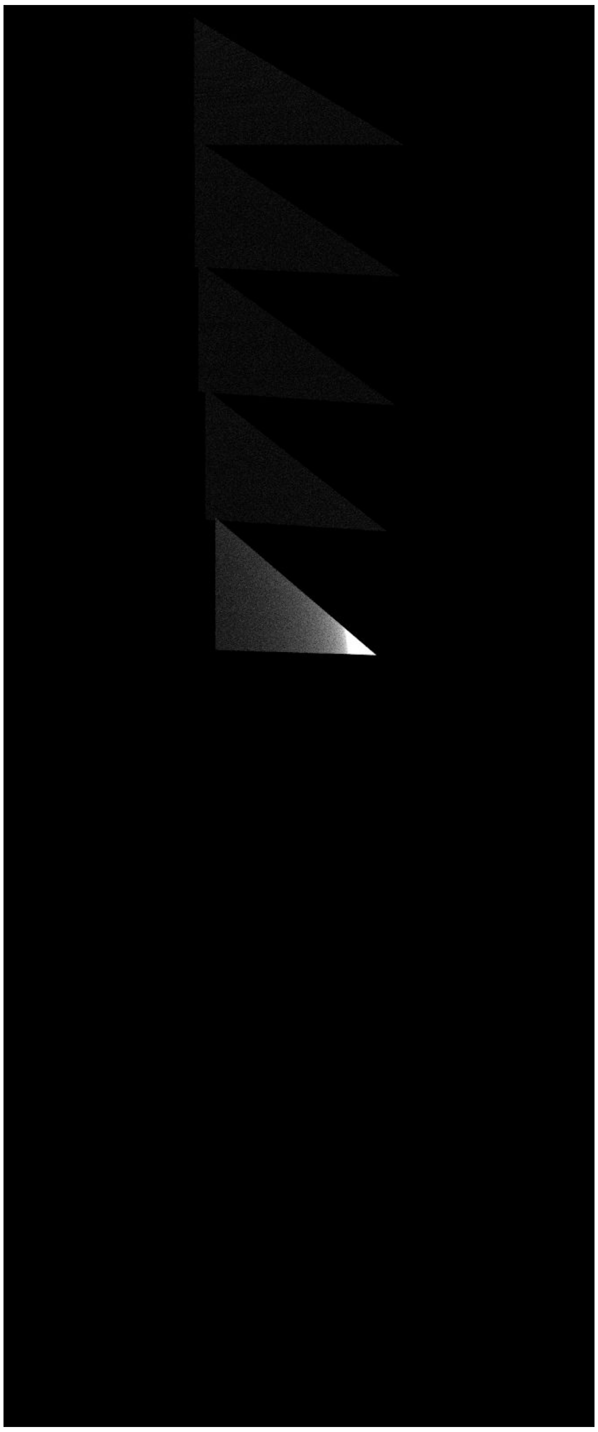

6. Diffuse material, mattness = 0.4


7. Diffuse material, mattness = 0.2

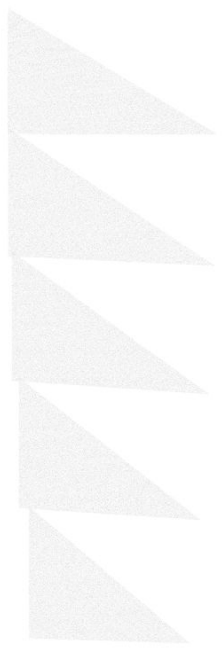

8. Diffuse material, mattness = 0



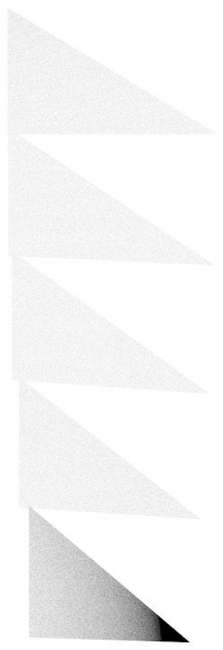
9. Transparent material, refractive index 1,5, normal of the surface towards the viewer, light source behind the camera,



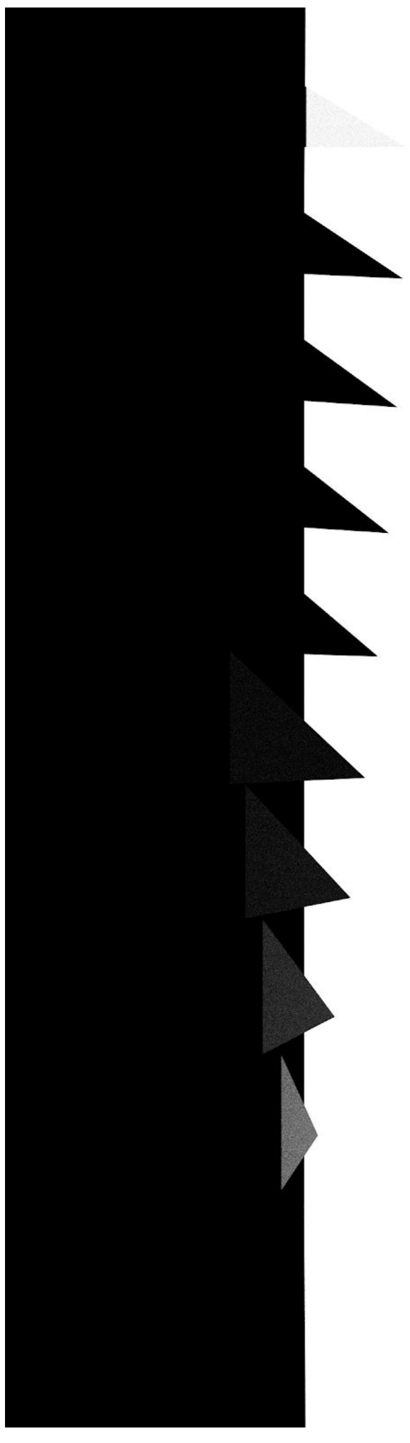
10. Transparent material, refractive index 1,5 normal of the surface away from the viewer, light source behind the camera,



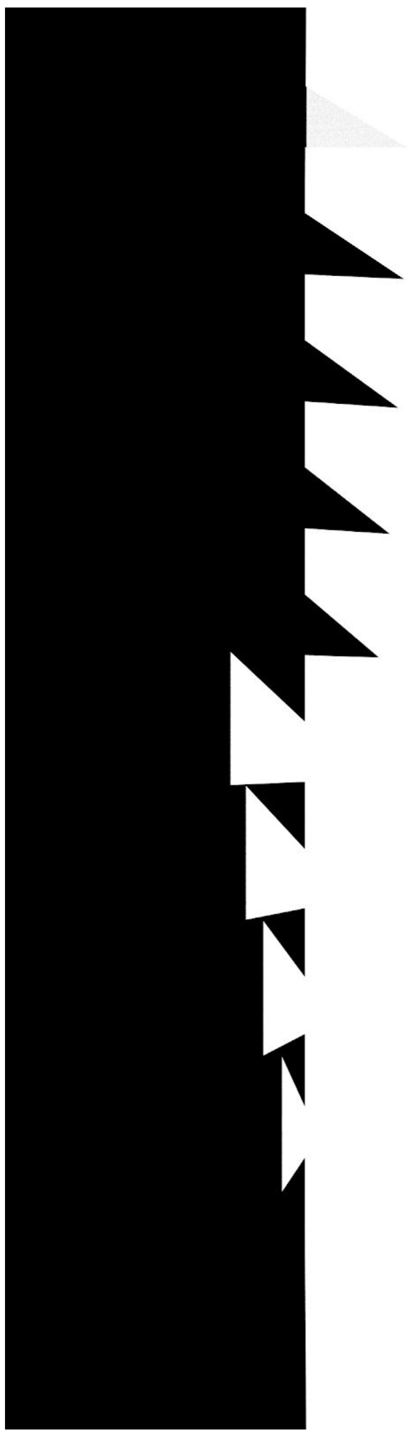
11.Transparent material, refractive index 1,5, normal of the surface towards the viewer, light source behind the triangles,



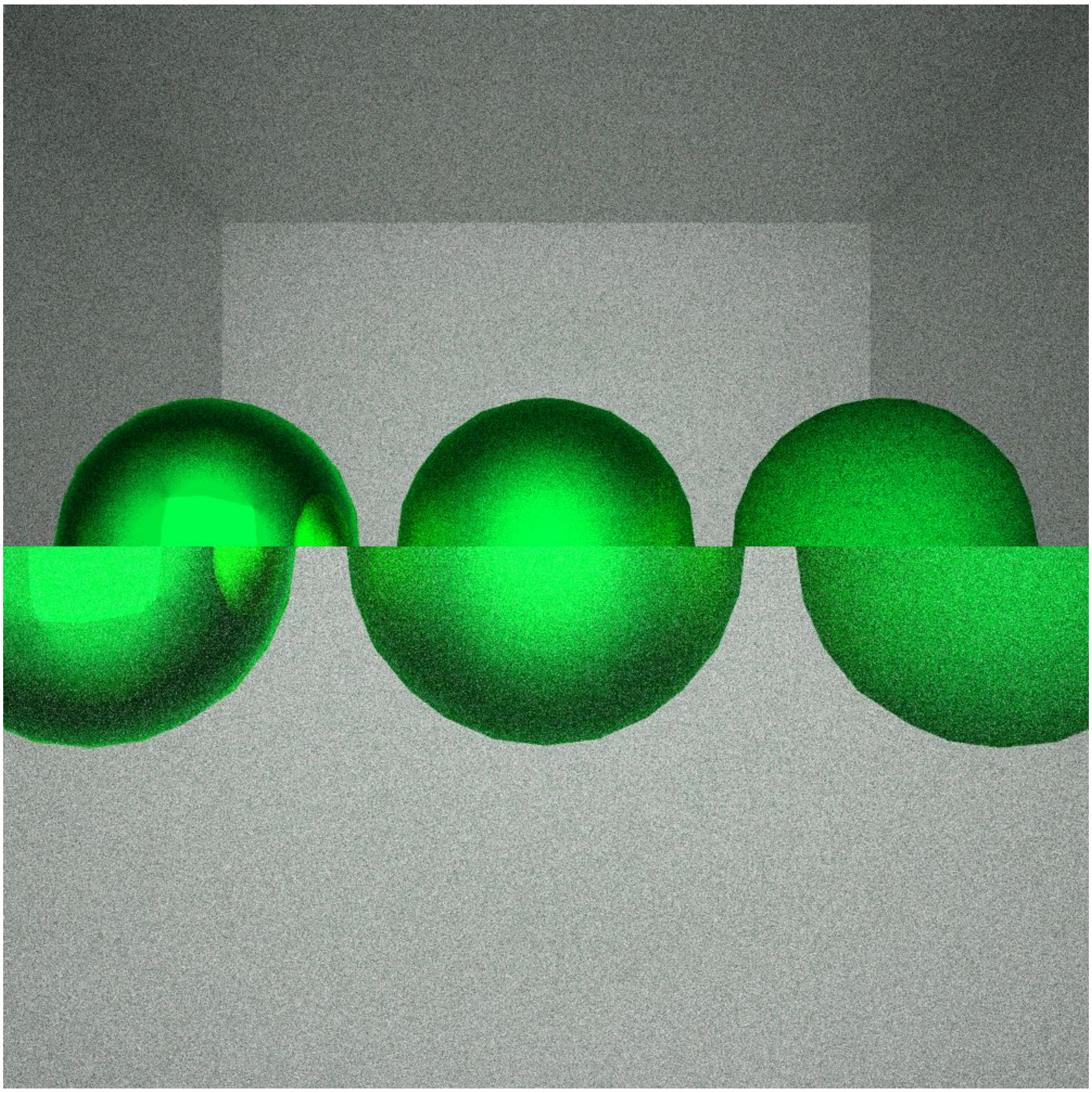
12. Transparent material, refractive index 1,5 normal of the surface away from the viewer, light source behind the triangles,



13. Transparent material, refractive index 1,5, normal of the surface towards the viewer, light source on the right side of the triangles



14. Transparent material, refractive index 1,5 normal of the surface away from the viewer, light source on the right side of the triangles


15. Glass panel in fron of demo 2. As can be seen, the image is incorrectly magnified behind the glass panel.

Class Interfaces

Body

Members

name	type	default value	description
material_	Material*		Pointer to the material of the body, the material is stored by Environment

Functions

name	arguments	return value	description
Body()	Material* material		Constructor
GetMaterial() const		const Material*	Returns material_
virtual Find collision()	Ray& ray	tuple<float, float, float>	
virtual Reflect()	Ray &ray, Vec3 &new_origin, float, float	void	

Camera

Members

name	type	default value	explanation
rays_	&Array(&Ray)		Reference to the array of references to rays which compose the picture.
n_rays_	size_t		Number of rays to shoot =x_resolution*y_resolution
x_resolution_	size_t	1000	width of picture in pixels
y_resolution	size_t	1000	height of picture in pixels
direction_	&Path	Path(Point(1,0,0), Vector(-1,0,0))	Where from and where at the camera is pointed
x_angle	spfp	45	width of picture in degrees
y_angle	spfp	45	height of picture in degrees

Functions

name	arguments	return type	explanation
Camera	(size_t x_resolution, size_t y_resolution, &Path direction, float x_angle, float y_angle)		Constructor, also creates an n_rays_ sized Array for rays
GetImage	(&Environment environment)	void	Calculates all of the rays, composes an image and saves it
SetAngles	(float x_angle, float y_angle)	void	Sets angles for the camera
SetDirection	(&Path)	void	Sets the direction for the camera
SetResolution	(size_t x_resolution, size_t y_resolution)	void	Sets resolution for the camera
PrintInfo	()	void	Prints current properties of the camera

Color

Members

name	type	default value	description
r_, g_, b_	float		RGB factors between 0 and 1

Functions

name	arguments	return value	description
Color()	float r, float g, float b		Constructor
Color()	tuple<float, float, float> rgb		Constructor
Color()	Color& c		Copy constructor
operator=	Color& c		Copy constructor
Color()	const Color& c		Copy constructor
operator=	const Color& c		Copy constructor

GetComponents() const		tuple<float, float, float>	Returns r_, g_ and b_
GetComponents255 () const		tuple<int, int, int>	Returns r_, g_ and b_ converted to RGB values

Common math operators +,* and / and printing operator << have been implemented for Color.

Environment

Members

name	type	default value	description
bodies_	vector<Triangle>		List of the bodies
materials_	vector<Material*>		List of the materials of the bodies in Environment

Functions

name	arguments	return value	description
Environment()			Constructor
~Environment()			Destructor
GetBodies()		vector<Triangle>&	Returns a vector of references to the bodies
AddBody()	Triangle& body		Adds a body to the environment. Called by parser.
AddMaterial()	Material* material		Adds a material to the environment. Called by parser.
MaterialAt()	int i	Material*	Returns the material at index i
PrintInfo() const		void	Prints the number of bodies to std output
LoadMesh()	std::string dir, std::string filename, Material* material, Vec3 midpoint, float height, float xrot,	void	Parses the .obj file mesh to Triangles

	float yrot, float zrot		
Count()	string str, char c	void	Counts number of c in str
BoundingBox()	std::vector<Triangle>& triangles	std::tuple<Vec3, Vec3>	Returns the max and min coordinates of a triangle

Image

Members

name	type	default value	description
colors_	std::vector<Color>		Vector of the colors for the pixels of the image
height_	int		Number of pixel rows in the image
width	int		Number of pixel columns in the image

Functions

name	arguments	return value	description
Image()	int height, int width		Constructor
GetPtrToPixel()	int i, int j	Color*	Returns pointer to the pixel at ith row and jth column
DrawPpm() const		void	Draws a ppm image
DrawPng() const		void	Draws a png image
Normalize()		void	Normalizes the pixels, so that the highest value of color component among the pixels becomes 1

Material

Members

name	type	default value	description
luminous_	bool		Whether material is radiant or not
color_	Color		Color of material

Functions

name	arguments	default value	description
Material()	bool luminous, Color color, Color color_rem		Constructor
GetLuminosity()		bool	Returns luminous_
GetColor()		Color	Returns color_
GetColorRem()		const Color	Returns color_rem_
virtual Reflect()	Vec3, Vec3	Vec3	

MaterialDiffuse

Members

name	type	default value	description
luminous_	bool	false	
color_	Color		
matness_	float		Describes how much the material deflects the path of the ray compared to specular reflection, between 0 and 1

Functions

name	arguments	default value	description
MaterialDiffuse()	bool luminous, Color color, Color color_rem, float mattness		Constructor
MaterialDiffuse()	MaterialDiffuse& m		Copy constructor

MaterialDiffuse()	const MaterialDiffuse& m		Copy constructor
operator=	MaterialDiffuse& m		Copy constructor
operator=	const MaterialDiffuse& m		Copy constructor
Reflect()	Vec3 original_direction, Vec3 unit_normal	Vec3	Returns the new direction for the ray

MaterialSpecular

Members

name	type	default value	description
luminous_	bool	false	
color_	Color		

Functions

name	arguments	default value	description
MaterialSpecular()	bool luminous, Color color, Color color_rem, float mattness		Constructor
MaterialSpecular()	MaterialSpecular& m		Copy constructor
MaterialSpecular()	const MaterialSpecular& m		Copy constructor
operator=	MaterialSpecular& m		Copy constructor
operator=	const MaterialSpecular& m		Copy constructor
Reflect()	Vec3 original_direction, Vec3 unit_normal	Vec3	Returns the new direction for the ray

Ray

Members

name	type	default value	description
origin_	Vec3		The initial point of ray’s current trajectory
direction_	Vec3		The current direction of the ray
color_	Color		Stores color factors from collisions
finished_	bool	false	Is set true, if ray has collided with a light source or if its current path does not coincide with any bodies. Set by camera.

Functions

name	arguments	return value	description
Ray()	Vec3 origin, Vec3 direction, Color color		Constructor. If arguments are not provided, constructs a ray with zero vectors for origin and direction and (1,1,1) for color
SetNewDirection()	Vec3 dir	void	Sets dir as the direction of the vector, called by Camera
SetNewOrigin()	Vec3 orig	void	Sets orig as the origin of the vector
SetNewColor()	Color ncolor	void	Sets ncolor as the color of the vector, called by Camera
RemoveColor()	Color k	void	Changes the components color of the vector by the factors in k

SetFinished()		void	Sets finished_ to true
GetColor()		Color	Returns color_
GetDirection()		Vec3	Returns direction_
GetOrigin()		Vec3	Returns origin_
IsFinished		bool	Returns finished_

Scene

Functions

name	arguments	return value	description
Scene()			Constructor
DrawImageFrom()	std::string fileName, int samples_per_pixel	void	Parses the Environment, and calls GetImage() of Camera
ParseVector()	json &vector	Vec3	Parses a Vec3 from the .json file
ParseColor()	json &color	Color	Parses a Color from the .json file

Triangle

Members

name	type	default value	description
v0_, v1_, v2_	Vec3		Vertices of the triangle
n0_, n1_, n2	Vec3		Unit normals corresponding to each corner of the triangle
unit_normal_	Vec3		Unit normal of the

			triangle, in case there’s only one
using_vertex_normals	bool		Defines whether vertex normals are used

Functions

name	arguments	return value	description
Triangle()	Material* material, Vec3 v0, Vec3 v1, Vec3 v2		Constructor for triangle without vertex normals
Triangle()	Material* material, Vec3 v0, Vec3 v1, Vec3 v2, Vec3 n0, Vec3 n1, Vec3 n2		Constructor for triangle with vertex normals
GetVertices()		tuple<Vec3, Vec3, Vec3>	Returns the vertices of the triangle
FindCollision() const	Ray &ray	tuple<float, float, float>	Finds the collision point of ray, returns the distances of the point to two vertices and the distance to the collision point from the origin of ray. If the ray does not collide with the triangle, returns
Reflect()	Ray &ray, Vec3 &new_origin, float u, float v	void	Reflects the ray according to material_
GetNormalAt()	Vec3& collision point, float u, float v	Vec3	Gives unit normal at the input point, used for finding the Phong interpolation normal when using vertex normals
MoveBy()	Vec3& v	void	Moves the triangle by vector v
ScaleBy()	float c	void	Scales the triangle by c
RotateAroundXAxisBy()	float deg	void	Rotates the triangle around x-axis by deg degrees

RotateAroundYAxis By()	float deg	void	Rotates the triangle around y-axis by deg degrees
RotateAroundZAxis By()	float deg	void	Rotates the triangle around z-axis by deg degrees

Vec3

Members

name	type	default value	explanation
x_, y_, z_	float		vector components in the Cartesian coordinates

Functions

name	arguments	return type	explanation
Vec3()	float x, float y, float z		Constructor
Vec3()			Constructor, initializes a vector with value 0 for each vector component
Vec3()	Vec3& v		Copy constructor
Vec3()	const Vec3& v		Copy constructor for const Vec3
operator=	Vec3& v	Vec3	Changes the values of x_, y_ and z_ to the corresponding values of the input vector and returns this
operator=	const Vec3& v	Vec3	Changes the values of x_, y_ and z_ to the corresponding values of the input vector and returns this
X() const		float	Returns the value of

			x_
Y() const		float	Returns the value of y_
Z() const		float	Returns the value of z_
Normalize()		void	Normalizes the vector
DotProduct()	Vec3& v	float	Returns the dot product of the vectors
CrossProduct()	Vec3& v	Vec3	Returns the cross product of the vectors
Norm()		float	Returns the norm of the vector
Reverse() const		Vec3	Returns a vector with opposite direction
IsZeroVector()		bool	Returns true if all the components are 0, false otherwise
RotateAroundXAxis( )	float deg	Vec3	Rotates the vector around x-axis by deg degrees
RotateAroundYAxis( )	float deg	Vec3	Rotates the vector around y-axis by deg degrees
RotateAroundZAxis( )	float deg	Vec3	Rotates the vector around z-axis by deg degrees

Common math operators and a printing operator have been implemented for Vec3.

Namespace Random

Function

name	argument	return value	description
randomNumber()		float	returns random number that’s in [0, 1]

randomNumber()	float min, float max	float	returns random number that’s in [min, max]
RandomVector()		Vec3	returns random vector where X, Y, Z each are random numbers in [0, 1]
RandomVector()	float min, float max	Vec3	returns random vector where X, Y, Z each are random numbers in [min, max)
RandomVector()	Vec3 vector, float degree	Vec3	returns random vector atmost "degree" degrees away from given vector "vector"
RandomVectorInUnitSpehere()		Vec3	returns random vector from unit sphere
RandomVectorInHemisphere()	Vec3 normal	Vec3	returns random vector from the same hemisphere with given "normal" vector