Computer Graphics Project Report

Group 121

Members:

- Kirill Zhankov (5771455)
- Sorin Ciuntu (5755417)
- Razvan Dinu (5755123)

Basic Features

Bounding Volume Hierarchy

Splitting by median			
Scene	**Cornell Box (mirror)**	**Monkey**	**Dragon**
Num. of triangles	32	967	87130
Time to create (ms), average of 5	0.019	0.963	244
Time to render (ms), average of 5	145	146	155
BVH Levels	4	9	16
BVH Leaves	8	256	32768
SAH + Binning			
Time to create (ms), average of 5	0.083	3.710	645
Time to render (ms), average of 5	45	77	91

Sources:

- Centroid of a triangle: *Protter, Murray H.; Morrey, Charles B. Jr. (1970), College Calculus with Analytic Geometry (2nd ed.), Reading: Addison-Wesley, LCCN 76087042, p. 520*

- Sorting function in standard library*:* *https://www.digitalocean.com/community/tutorials/sort-in-c-plus-plus*
- Finding the height of a binary tree iteratively: *https://www.geeksforgeeks.org/iterative-method-to-find-height-of-binary-tree/*

Shading Models

Sources:

- Sorting a vector of pairs using a comparator function: https://www.geeksforgeeks.org/sort-vector-of-pairs-in-ascending-order-in-c/

Lights and Shadows

Incorrectness in visibilityOfLightSampleTransparency():

Firstly, the operation treats blending like it is commutative. In reality, having first a red, then a yellow glass between the point and the light leads to a different perceived color than having first a yellow, and then a red one.

Secondly, it doesn't use Beer's Law. A light ray traveling through a thicker material should lose more intensity, but in this case it does not.

Course slides showing the importance of order when alpha-blending, which also applies here, but is not taken into account by the given formula.

Sources:

- Parallelogram Light: *Marschner, S.; Shirley, P. Fundamentals of Computer Graphics, Fourth.; CRC Press, Taylor & Francis Group: Boca Raton, FL, 2015, chapter 13.4.2.*

- Bilinear Interpolation: *https://en.wikipedia.org/wiki/Bilinear_interpolation#Inverse_and_generalization*
- Beer's Law: *Marschner, S.; Shirley, P. Fundamentals of Computer Graphics, Fourth.; CRC Press, Taylor & Francis Group: Boca Raton, FL, 2015, chapter 13.1.*
- Importance of order in blending: Course Slides

Extra Features

SAH + Binning

The table with timings can be found in the BVH section.

1. 50 bins - a good balance between construction and rendering time.
2. Assign primitives to bins.
3. 1.5 for traversal cost showed good performance.
4. Find splitting with the lowest cost; also compare minCost with baseCost to determine if the split is worth it.

I visualized selected splitting at the selected node. Optimal splittings are shown with filled boxes; remaining splittings are shown in wireframe mode. A class SplitInfo was added to store necessary information.

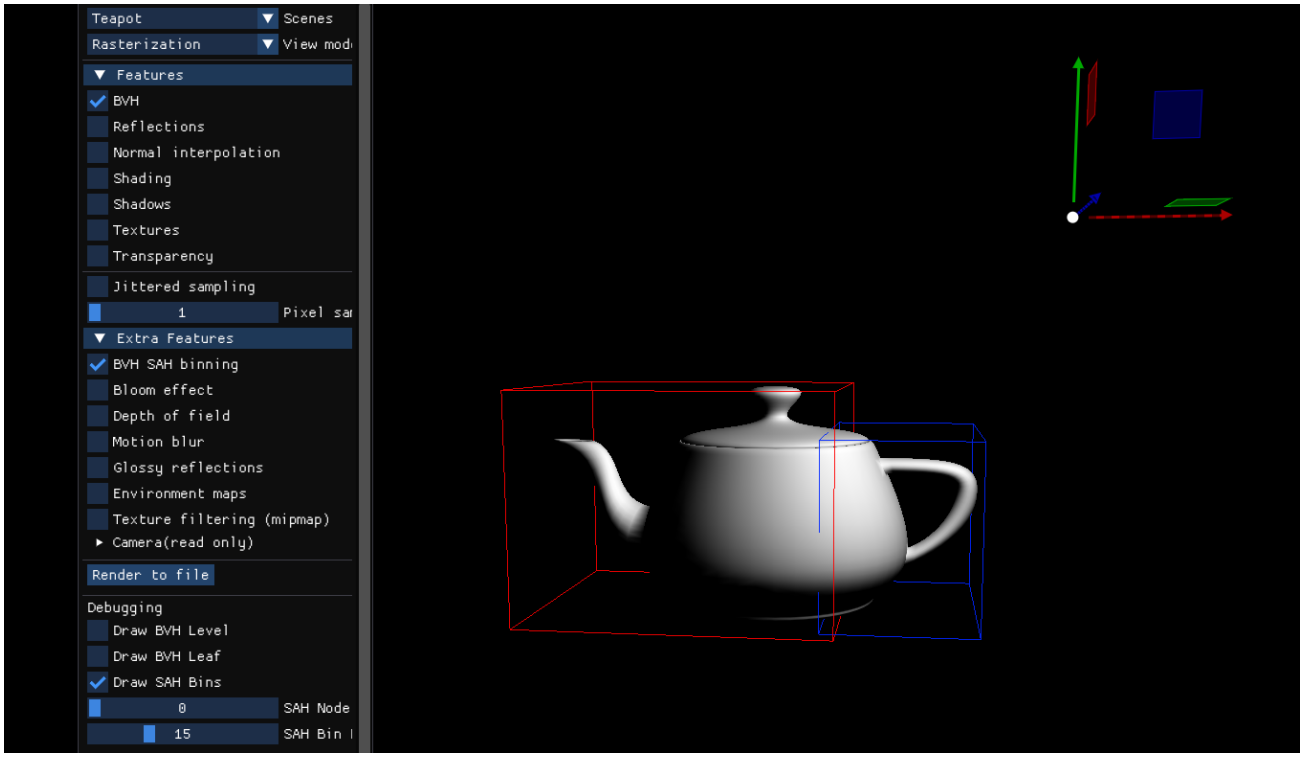


Image. Non-optimal split with SAH+Binning.

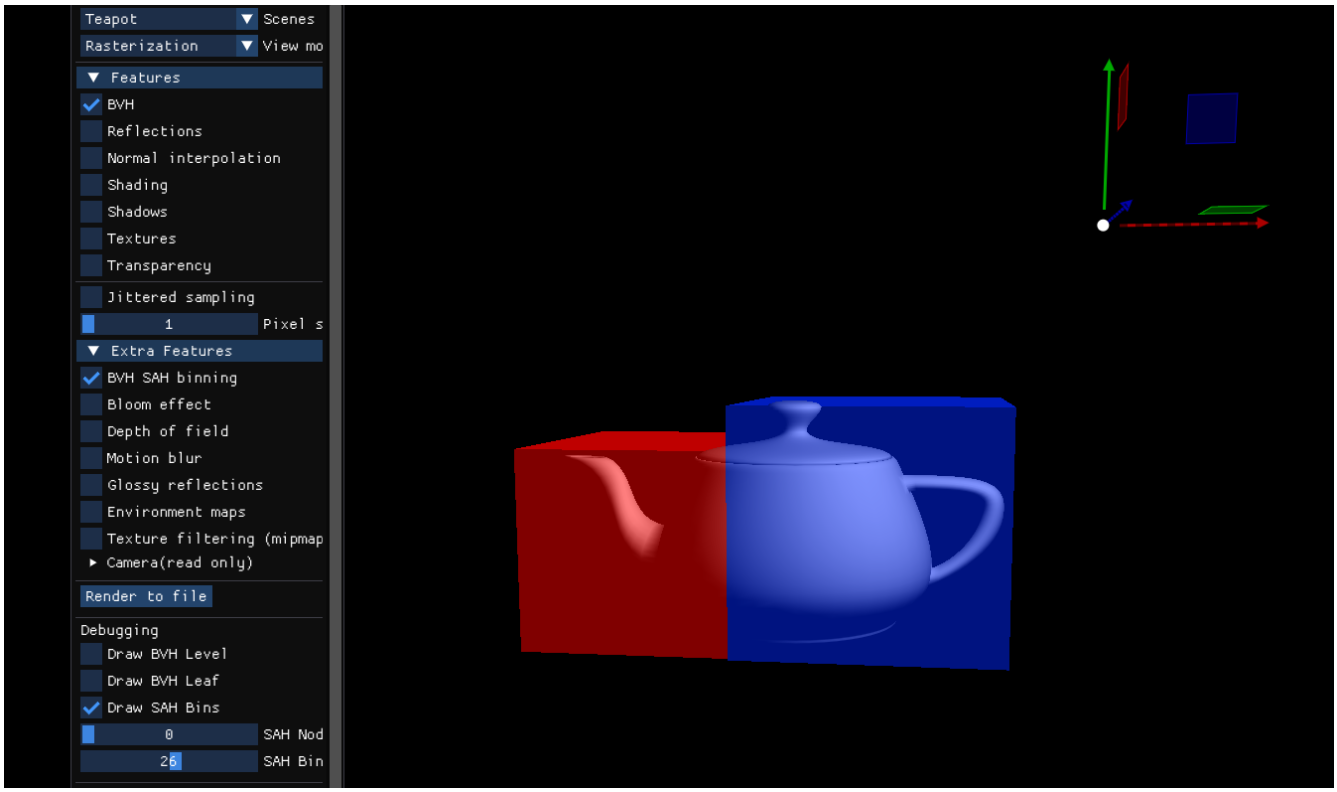


Image. Optimal split with SAH+Binning.

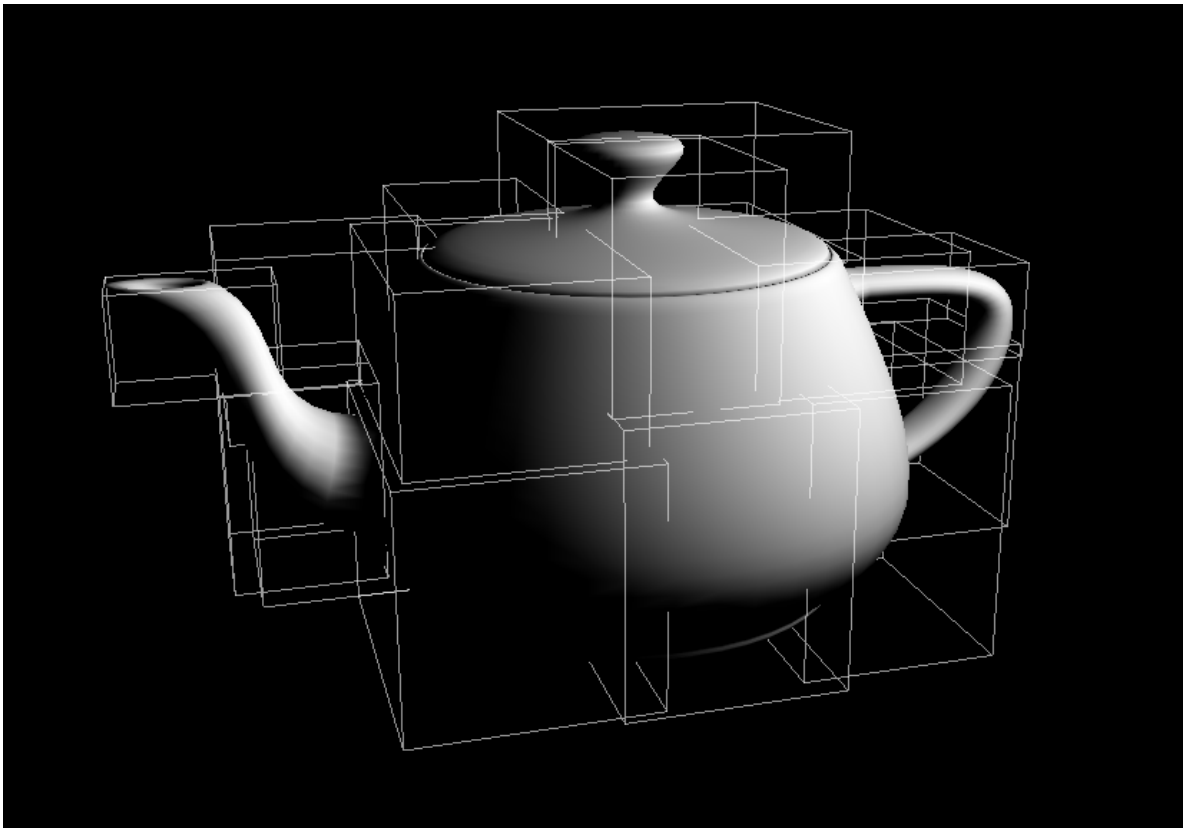


Image. Nodes of a tree created with SAH+Binning.

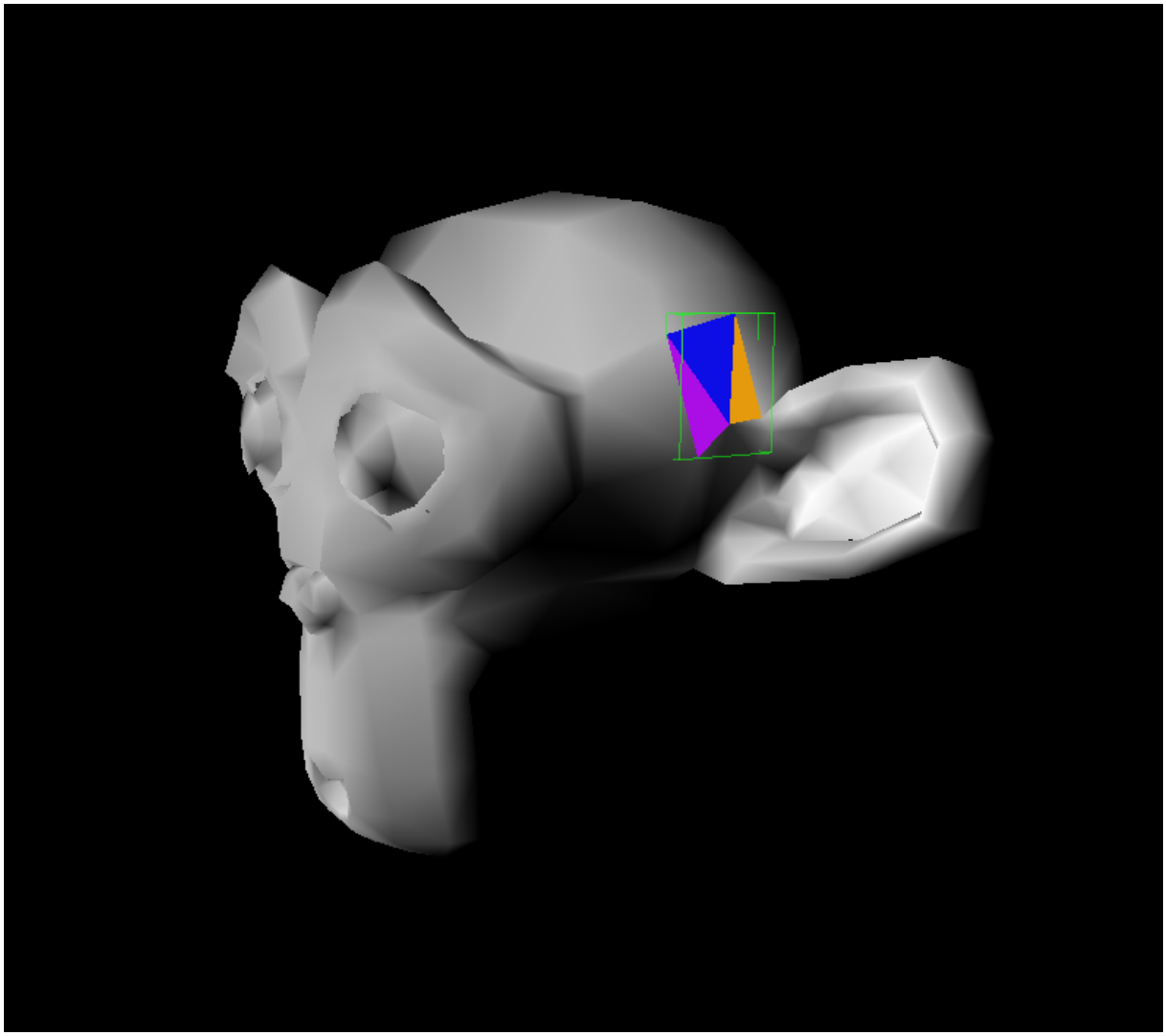


Image. Leaf of a tree created with SAH+Binning.

Locations:

- determineBucketIndex in extra.cpp
- calculateAABBSurfaceArea in extra.cpp
- splitPrimitivesBySAHBin in extra.cpp
- buildRecursive in bvh.cpp
- numberOfBinsInNode in bvh.cpp
- debugSAHBins in bvh.cpp
- Added fields to bvh.h
- Modified common.h and main.cpp to control parameters in GUI

Sources:

- Surface area heuristic with binning: *M. Pharr, J. Wenzel, and G. Humphreys. Physically Based Rendering, Second Edition: From Theory To Implementation. Morgan Kaufmann Publishers Inc., 2nd edition, chapter 4.4.2.*
- General information*: TU Delft Computer Graphics course, lecture 9.*

Bloom Filter

1. Make a copy of the image.
2. Pad it.
3. Set all pixels below a threshold to 0.
4. Apply filter to first rows, then columns.
5. Combine the filtered image with the original image.

- To prevent overflow, I limit filterSize.
- The threshold is compared to “perceived luminance”.
- I also added an intensity parameter.
- Visual debug: one checkbox allows you to see all values above the threshold. The other shows how the filtered (blurred) image looks before combining it with the original image.

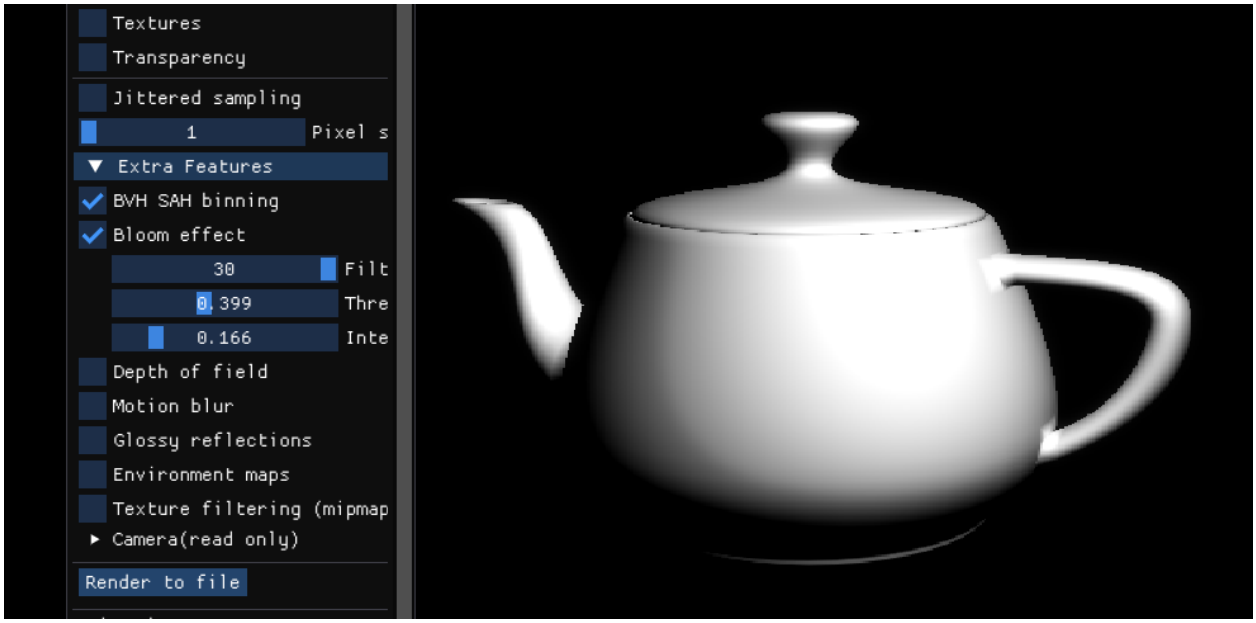


Image. Teapot with bloom effect.

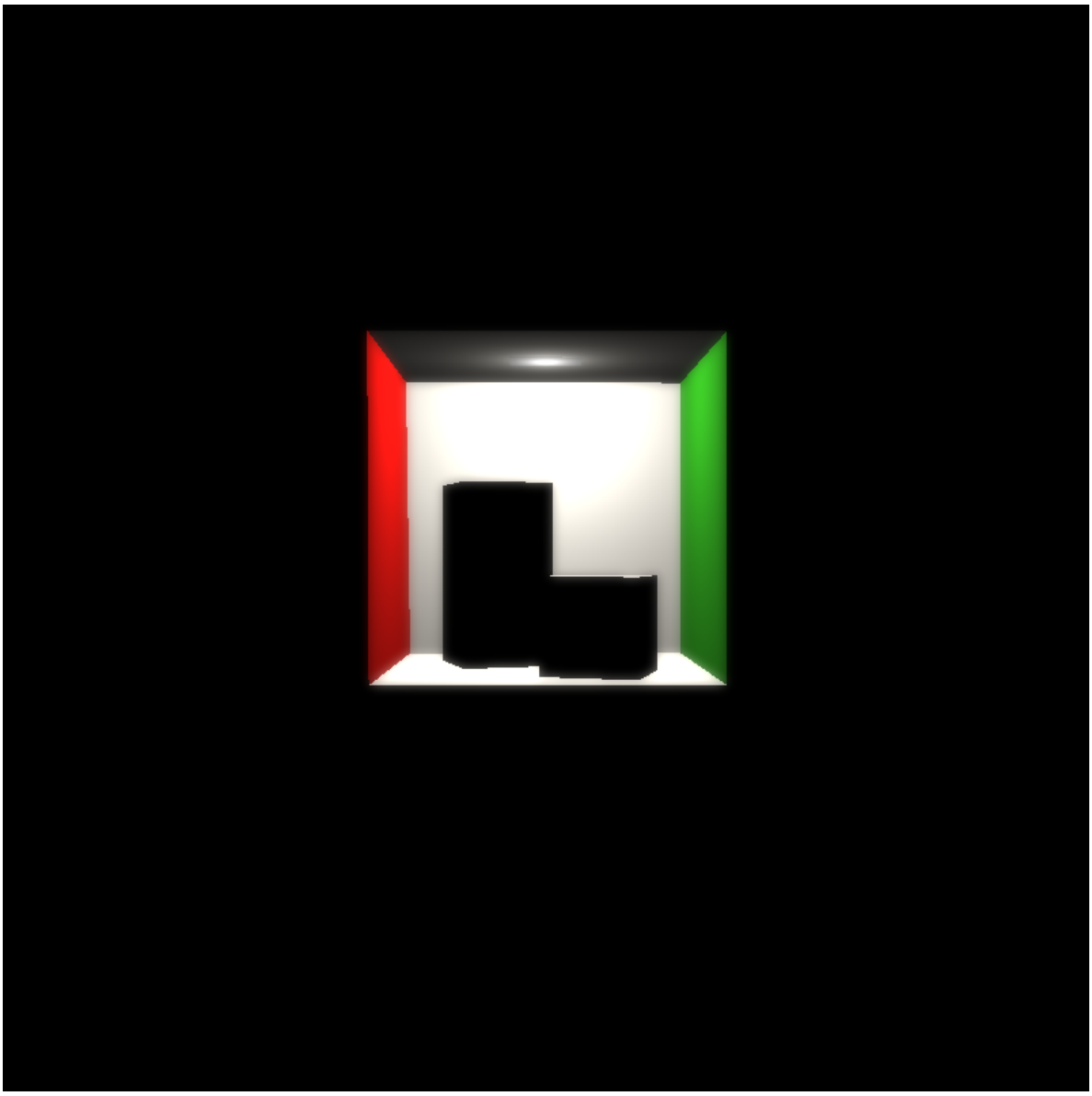


Image. Cornell box with bloom effect.

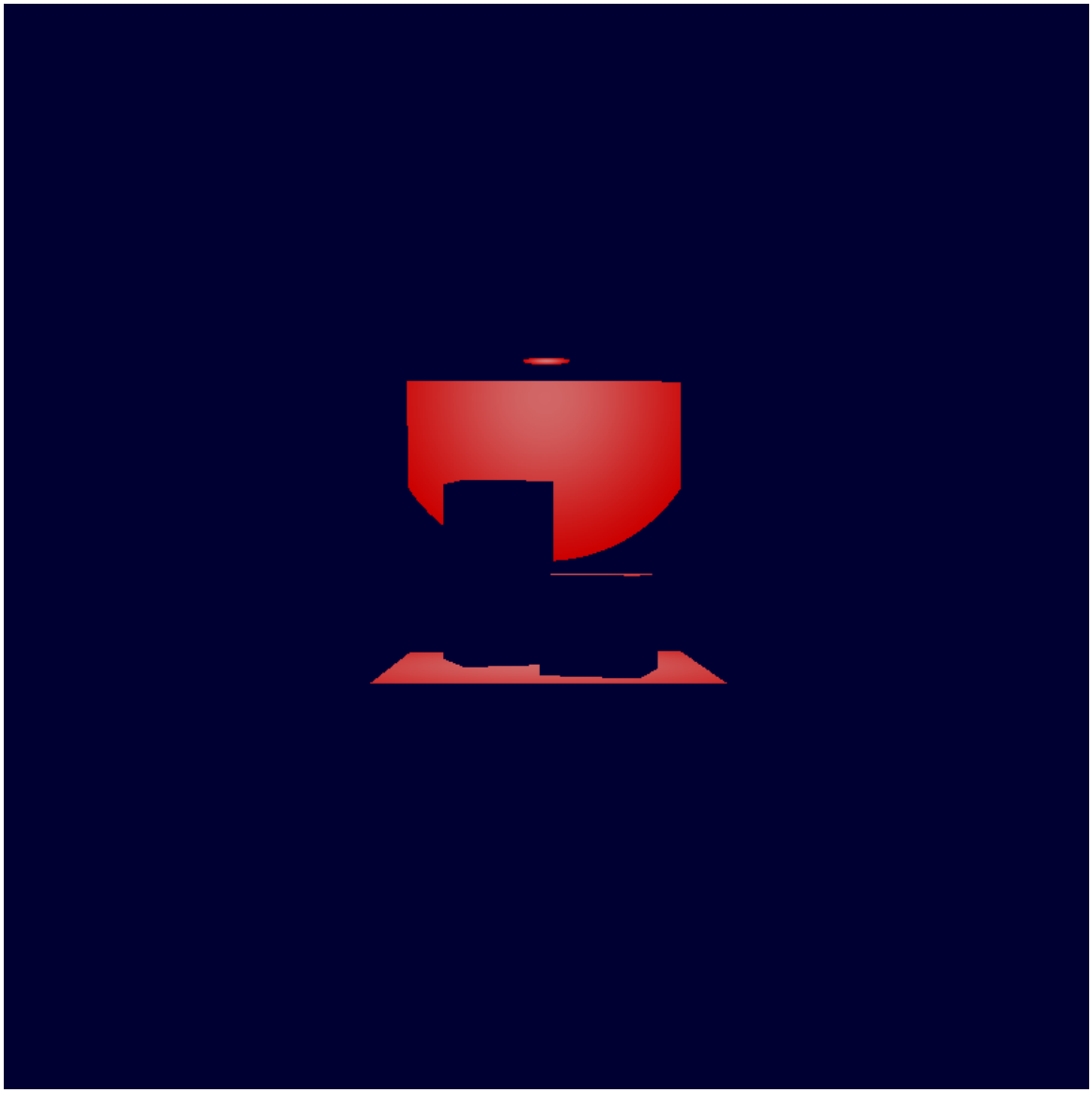


Image. Values above the threshold of bloom shown in red.

Image. Blur applied to values above threshold before combining images.

Locations:

- perceivedLuminance in extra.cpp
- applyThreshold in extra.cpp
- renderBloomAboveThreshold in extra.cpp
- renderBloomBlurredMask in extra.cpp
- padBorders in extra.cpp
- binomial in extra.cpp
- generateGaussianFilter in extra.cpp
- applyFilter1dToPixel in extra.cpp
- postprocessImageWithBloom in extra.cpp

- Modified common.h and main.cpp to control parameters in GUI

Sources:

- Perceived luminance: *https://stackoverflow.com/a/596243/15236567*
- Multiplicative formula for binomial coefficients: *https://stackoverflow.com/a/15302394/15236567*
- Convolution filters: *Marschner, S.; Shirley, P. Fundamentals of Computer Graphics, Fourth.; CRC Press, Taylor & Francis Group: Boca Raton, FL, 2015, p. 190*
- Binomial filters: http://www.cse.yorku.ca/~kosta/CompVis_Notes/binomial_filters.pdf.old

Glossy Reflections

1. Randomly sample disk lying in the plane orthogonal to reflected ray to obtain a point.
2. That point is the new vector tail.
3. Make sure the ray makes an acute angle with the normal.
4. Do that numGlossySamples times.

- Radius is inversely proportional to shininess.
- To keep the distribution uniform, we must take the square root of the random number.
- I show the reflected ray, perturbed rays and the disk represented by a sphere.

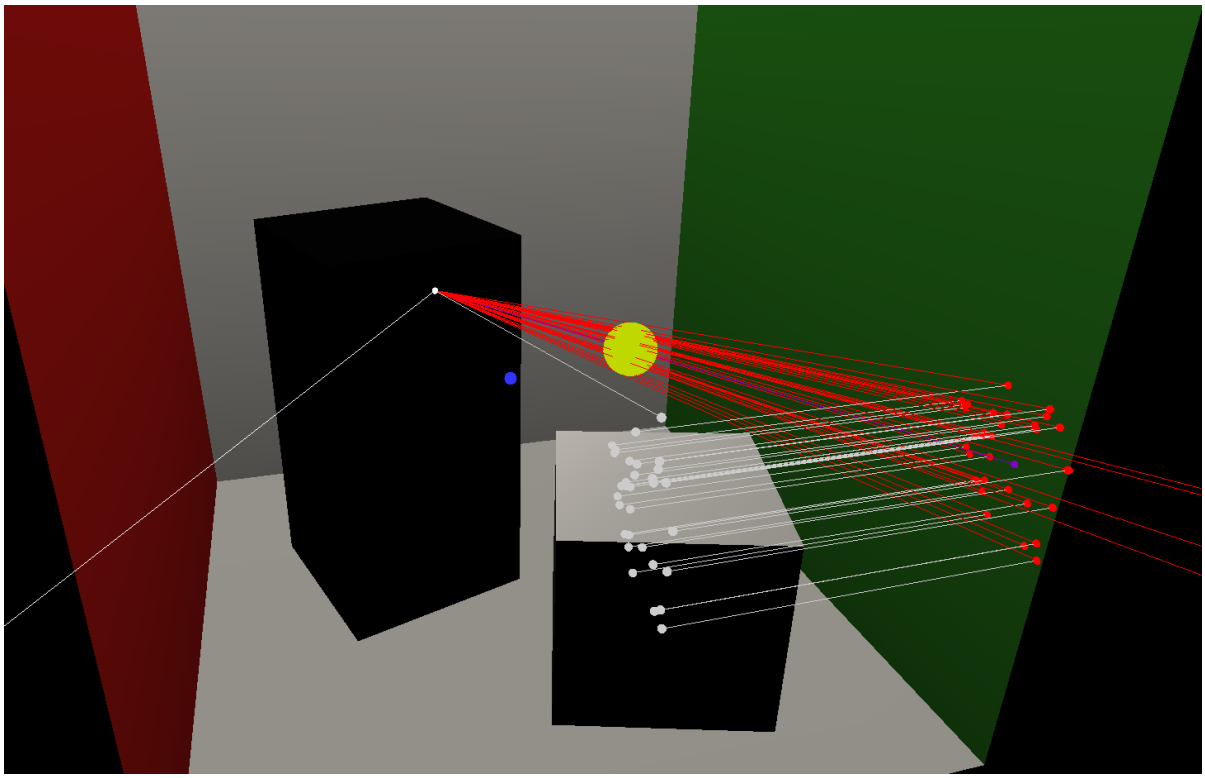


Image. Red perturbed rays bounded by the yellow sphere as well as the blue reflected ray.

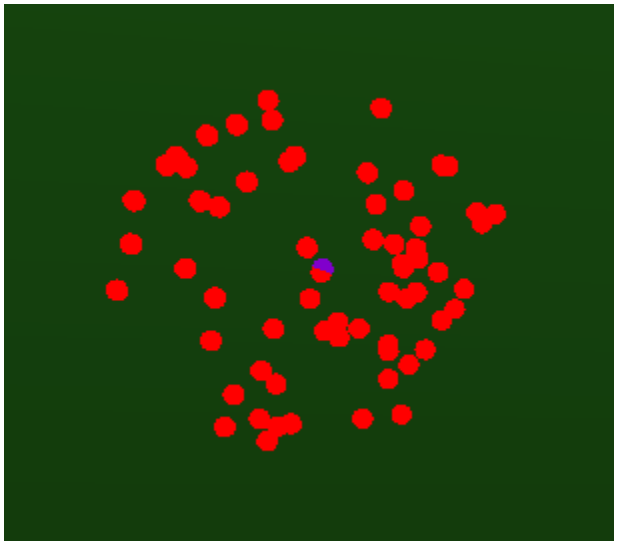


Image. Dots are distributed uniformly in a disk.

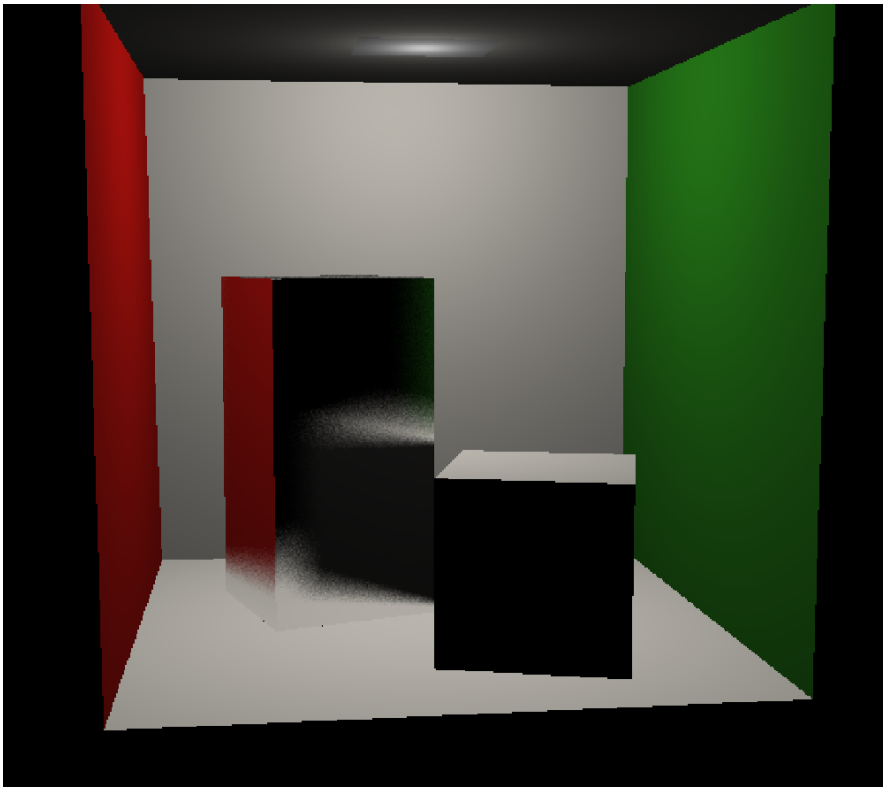


Image. Glossy reflections on.

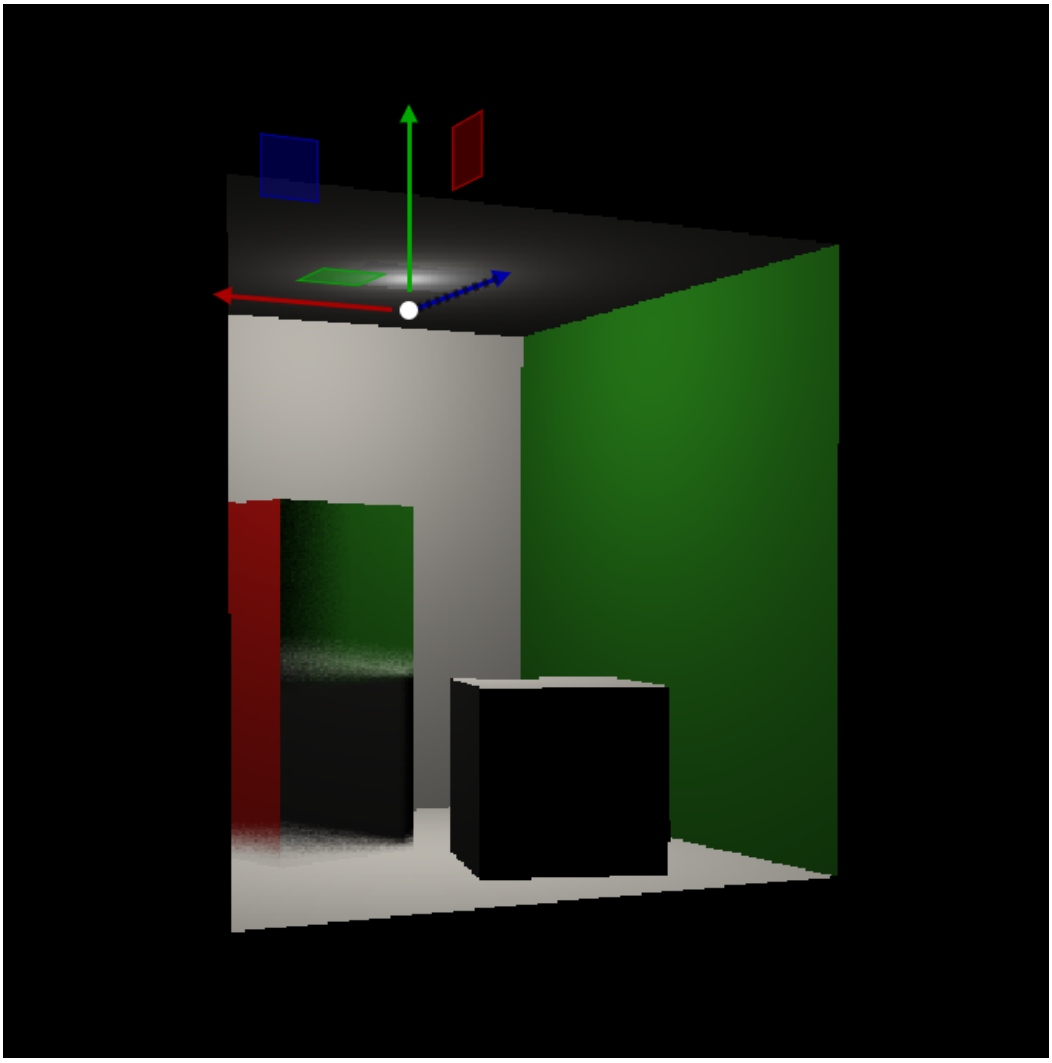


Image. Glossy reflections on.

Locations:

- constructOrthonormalBasis in extra.cpp
- sampleDisk in extra.cpp
- renderRayGlossyComponent in extra.cpp
- Modified common.h and main.cpp to control parameters in GUI

Sources:

- Sampling uniformly on a disk: *https://stackoverflow.com/a/50746409/15236567*
- Creating perturbed rays: *Marschner, S.; Shirley, P. Fundamentals of Computer Graphics, Fourth.; CRC Press, Taylor & Francis Group: Boca Raton, FL, 2015, chapter 13.4.4.*

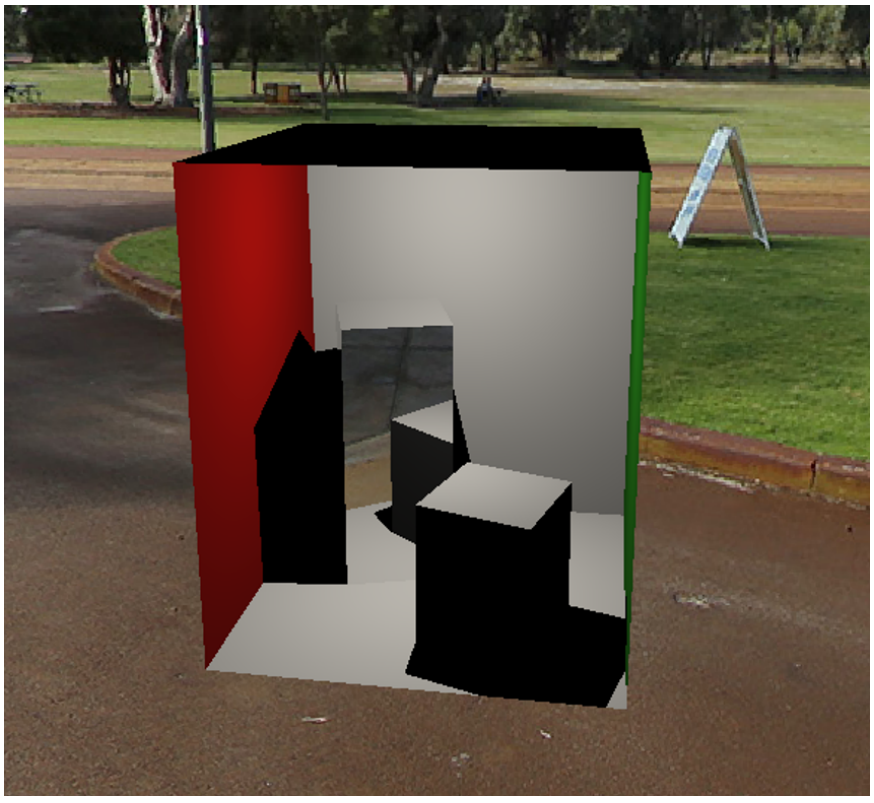
- Constructing an orthonormal basis from a single vector: *Marschner, S.; Shirley, P. Fundamentals of Computer Graphics, Fourth.; CRC Press, Taylor & Francis Group: Boca Raton, FL, 2015, chapter 2.4.6.*

Environment maps

- Implementation: cube map

Given the ray direction, we calculate which face of the cubemap it will hit by dividing all of the direction's components by the largest of these components in terms of its absolute value. This way, one component will indicate which face we should sample from, and the other two components represent the texture coordinates of the face we will sample from.

Environment mapping used with the ‘Dragon’ scene.



Environment mapping used with the ‘Cornell Box (with mirror)’ scene. It can be observed that the environment is reflected by the mirror.

Locations:

- extra.cpp (sampleEnvironmentMap())
- data/cube2.jpg (environment map texture)

- scene.h (added Image object)

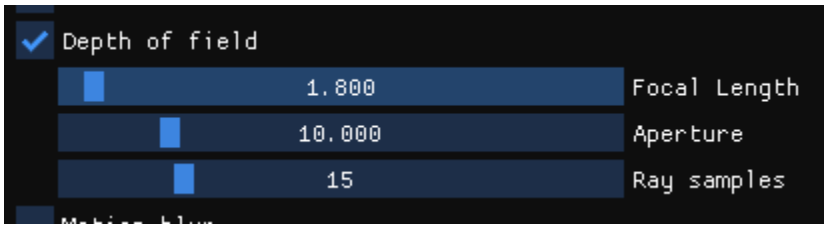
Sources:

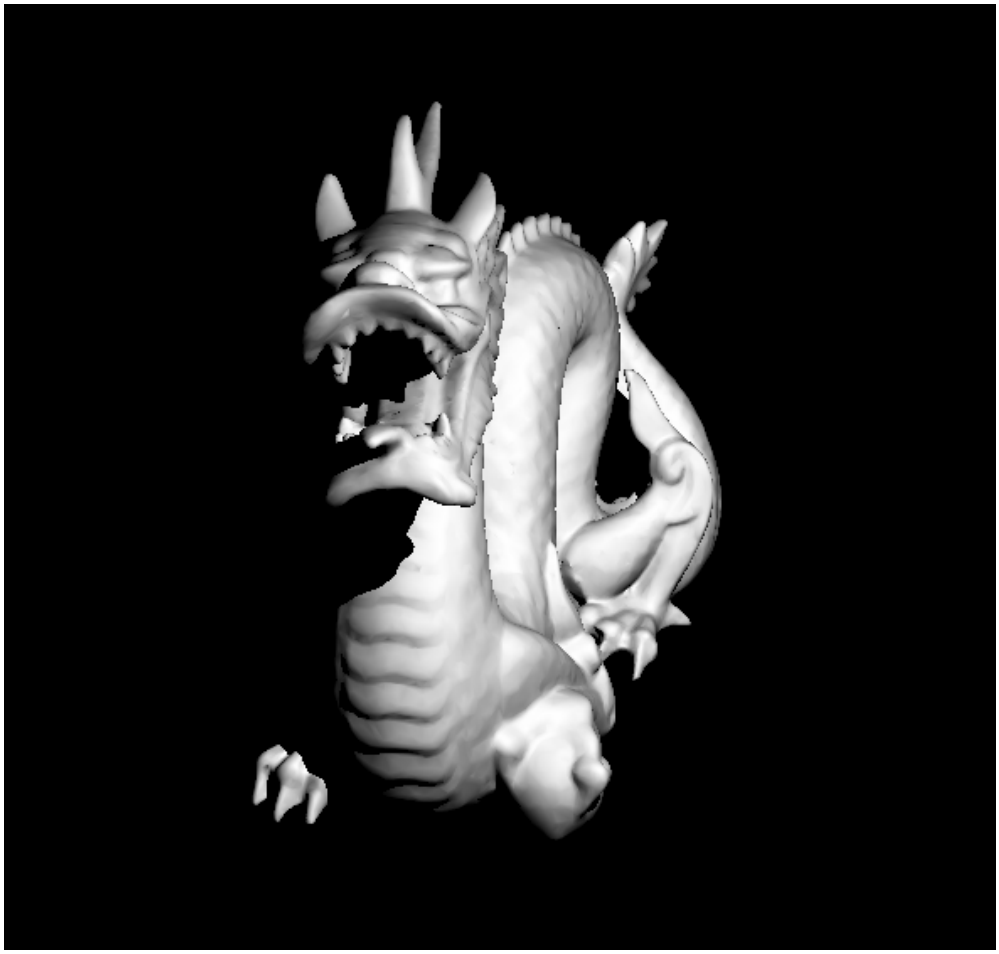
- *Marschner, S.; Shirley, P. Fundamentals of Computer Graphics, Fourth.; CRC Press, Taylor & Francis Group: Boca Raton, FL, 2015, chapter 11.4.5.*
- Scaling the direction vector: *McGill University Slides* - *https://www.cim.mcgill.ca/~langer/557/18-slides.pdf*

Depth of field

In order to simulate depth of field, we need to keep track of the focal length and the aperture of the supposed lens. The focal length determines where the rays should intersect with respect to the camera, while the aperture determines how big the area that we are shooting rays from is. All of the renders shown below have 25 ray samples per pixel.

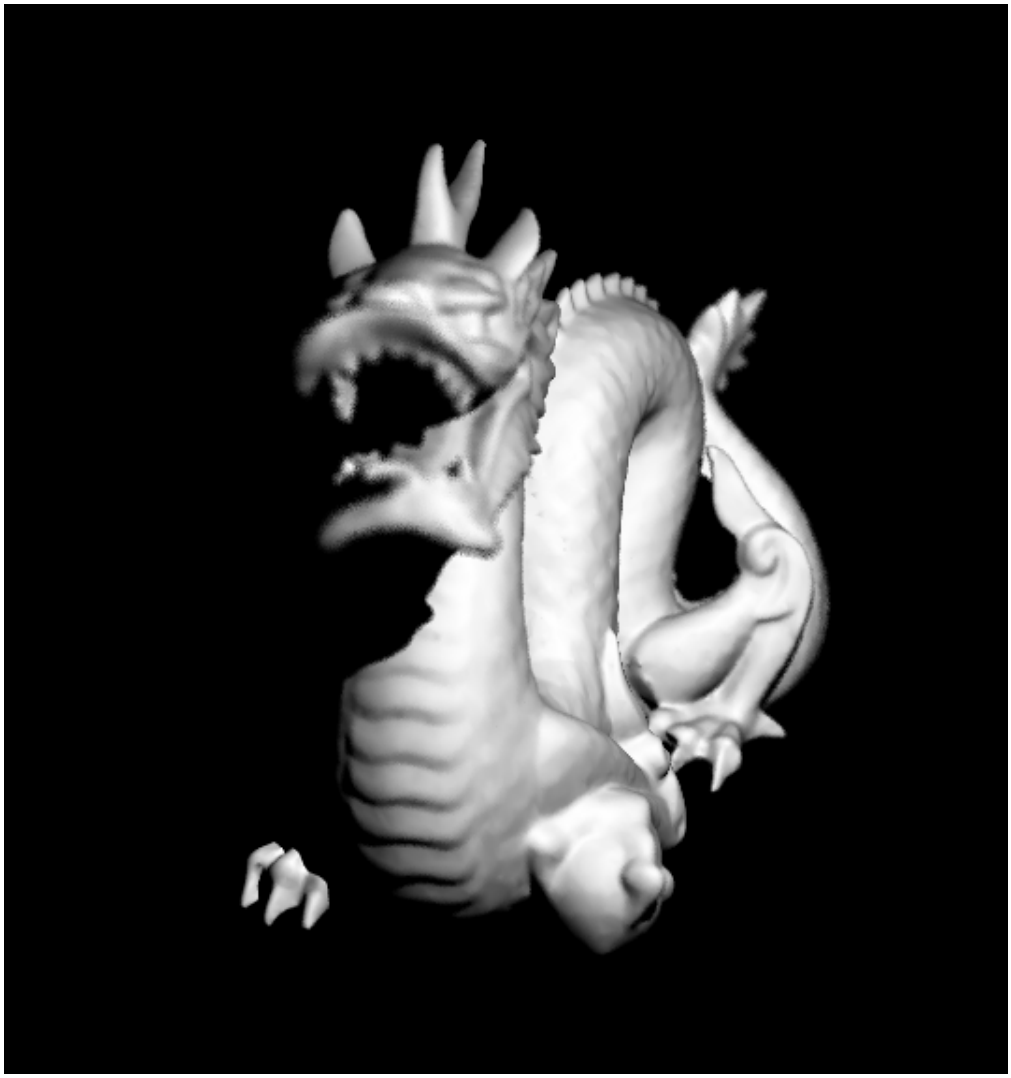
Visual debug: Once 'Depth of field' is ticked in the UI, 3 sliders will appear corresponding to the 3 variables discussed above:



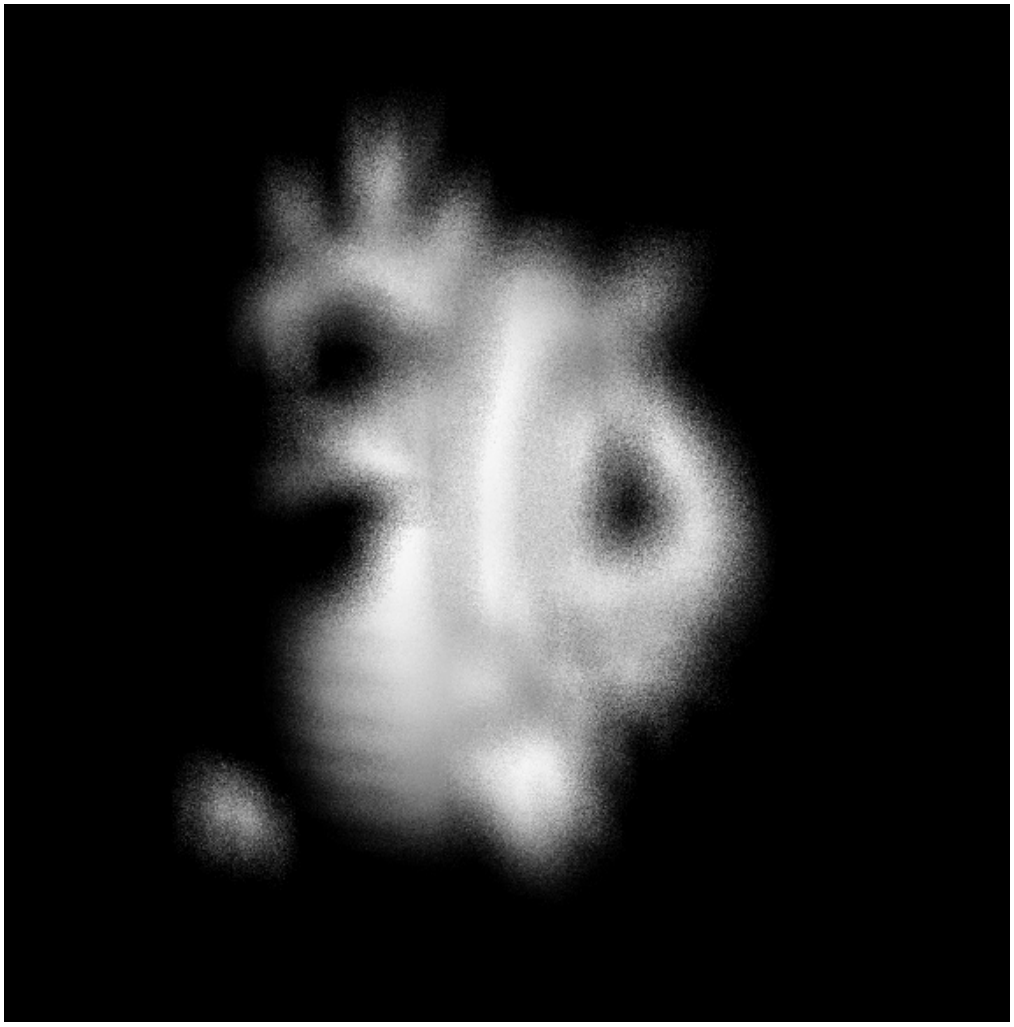


‘Dragon’ scene with depth of field turned off. (for reference)

‘Dragon’ scene with focal length 3 and aperture 30. The tail is slightly blurrier than the nose, since it is farther away from the focal plane.



‘Dragon’ scene with focal length 3 and aperture 50. An increase in aperture leads to a blurrier image.



‘Dragon’ scene with focal length 1 and aperture 50. By comparing with the previous image, this showcases how focal length plays a big role in how blurry the render is.

Locations:

- extra.cpp (renderImageWithDepthOfField())
- main.cpp (added sliders for focal length, aperture and number of samples, ~line 207)
- common.h (added variables in ExtraFeatures, corresponding to the 3 mentioned above)

Sources:

- *Marschner, S.; Shirley, P. Fundamentals of Computer Graphics, Fourth.; CRC Press, Taylor & Francis Group: Boca Raton, FL, 2015, chapter 13.4.3*

Motion Blur

Our implementation creates the motion blur effect using evenly distributed secondary scenes that simulate the movement of the objects across a given trajectory at equidistant intervals of time.

Each of these secondary scenes (samples) begin as a copy of the initial scene. They are then displaced on a trajectory given by the curve of a third degree polynomial currently coded to be: y = -4 * x^3 + 0.5 * x^2 + 1.5 * x, with x in [0,1). We then take the average of all samples to obtain our final render with the motion blur effect.

The visual debug option helps the user isolate individual samples by rendering only the desired sample at a time.

Images:

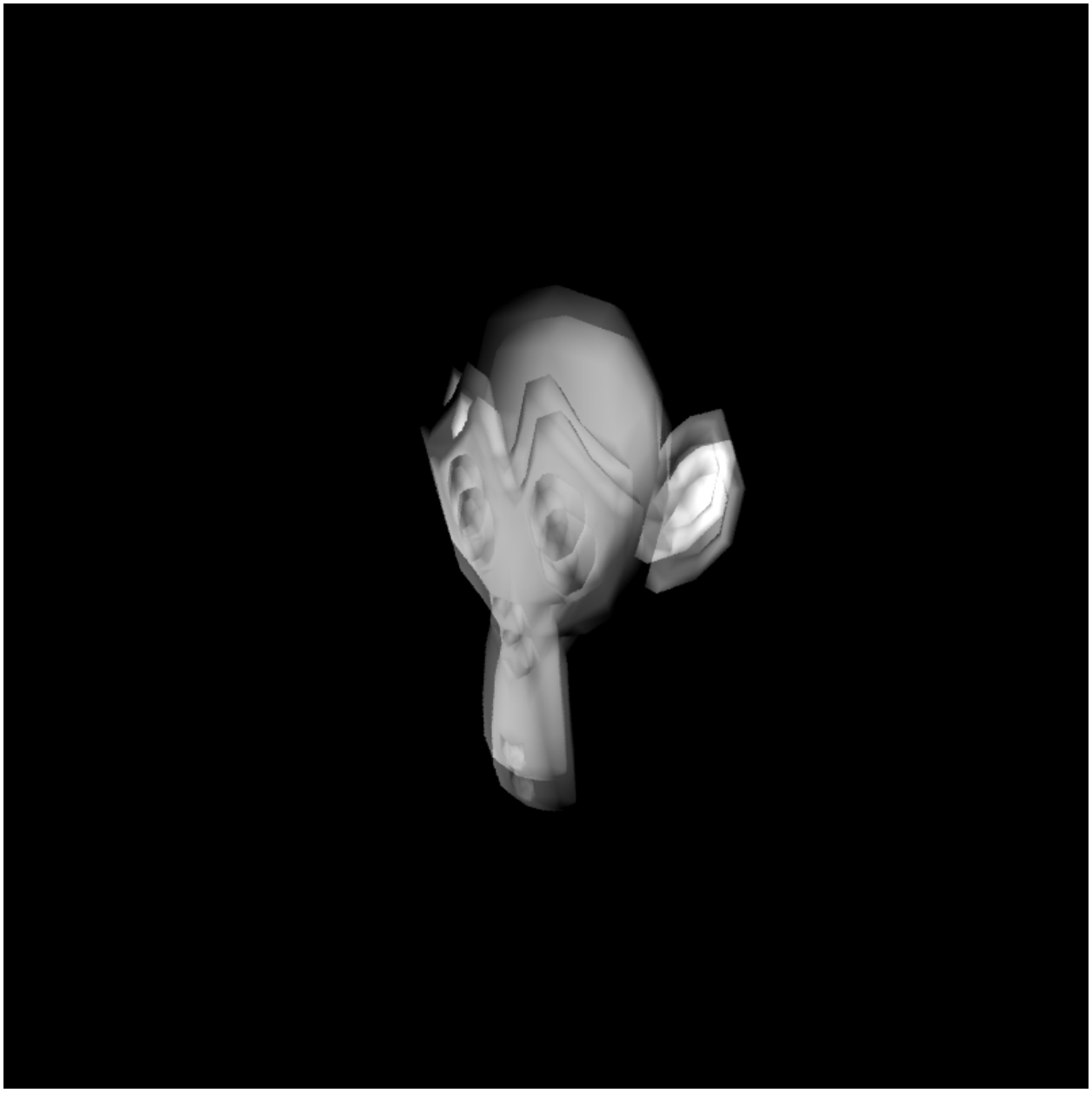
- All pictures taken from the default angle and zoom settings
- 3 renders: motion blur with 2 samples, 20 samples, 64 samples (with Lambertian shading + normal interpolation)
- 3 debug screenshots: isolated samples #1, #64, #500 (500 is outside normal bounds, however it helps illustrate the trajectory of the mesh in space) (max samples = 64, with Lambertian shading + normal interpolation)

Code location:

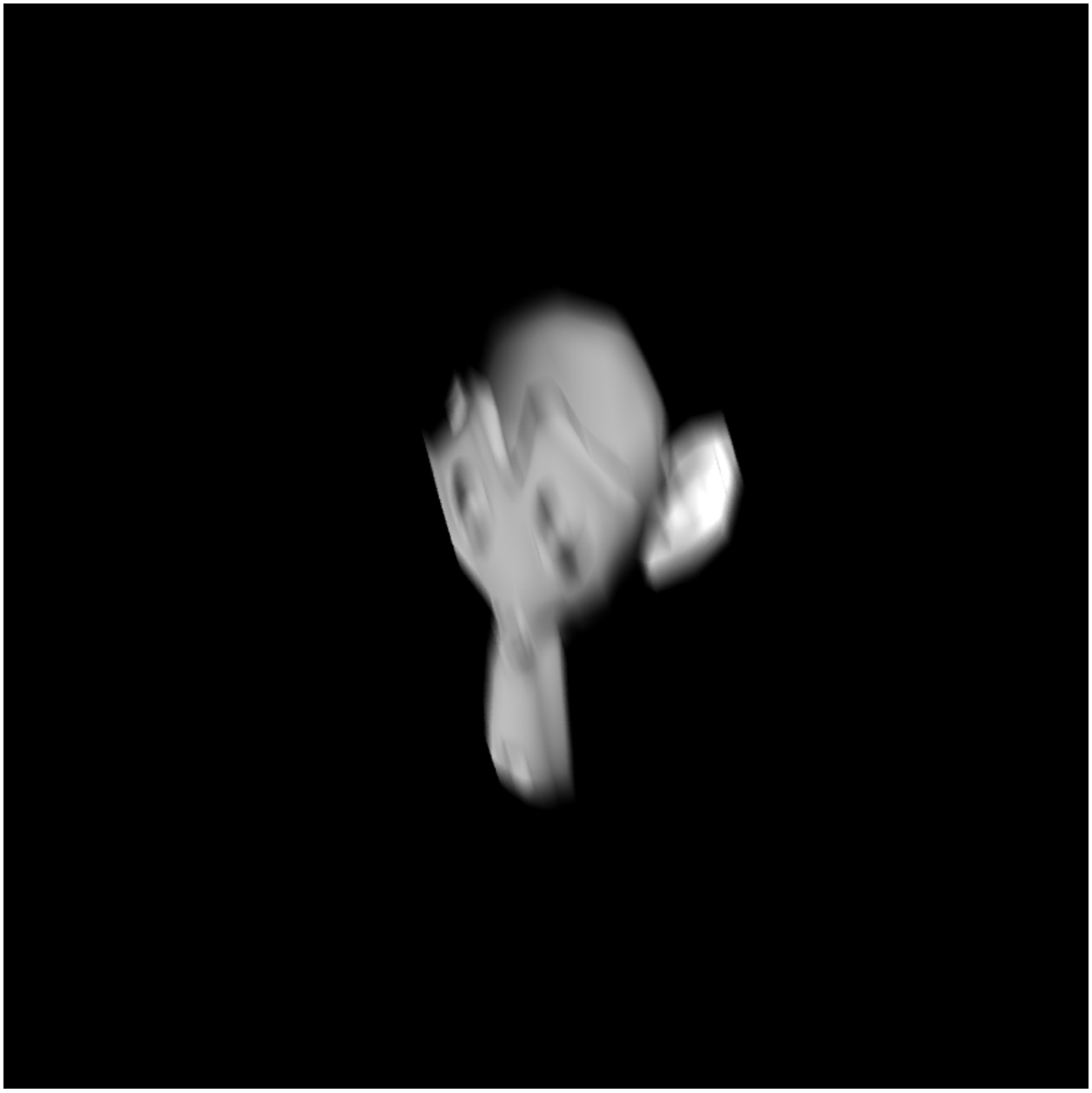
- extra.cpp - feature logic: renderImageWithMotionBlur()
- common.h - flags: numMotionBlurSamples, enableMotionBlurSampleIsolation, numMotionBlurSampleIsolated
- main.cpp - main function: GUI checkboxes and slider for aforementioned flags

Sources:

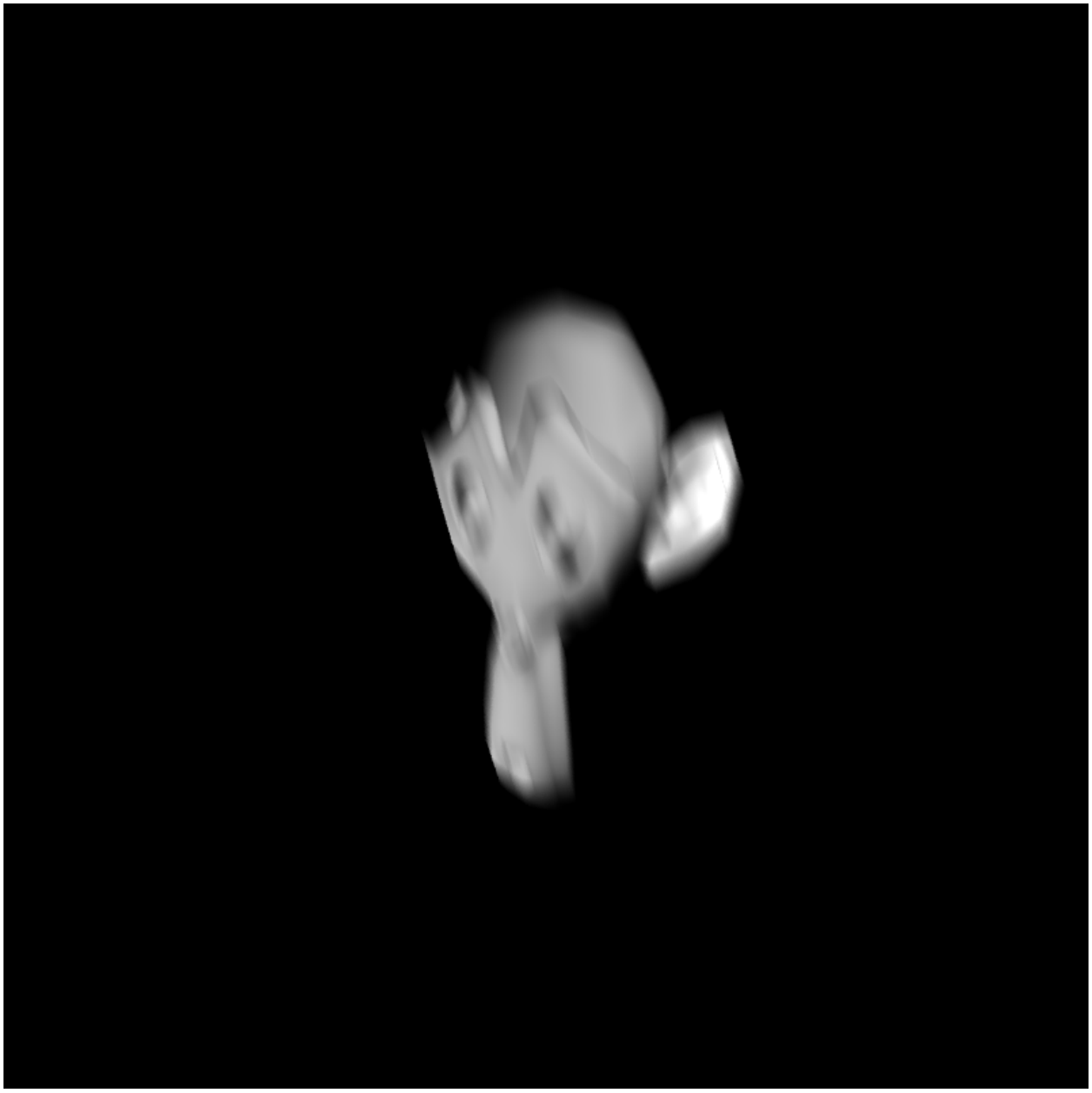
- Marschner, S., & Shirley, P. (2015). Fundamentals of computer graphics (Fourth). CRC Press, Taylor & Francis Group. (chapter 13.4.5 Motion Blur)
- Peter Shirley, Trevor David Black, Steve Hollasch. Ray Tracing: The Next Week. https://raytracing.github.io/books/RayTracingTheNextWeek.html#motionblur



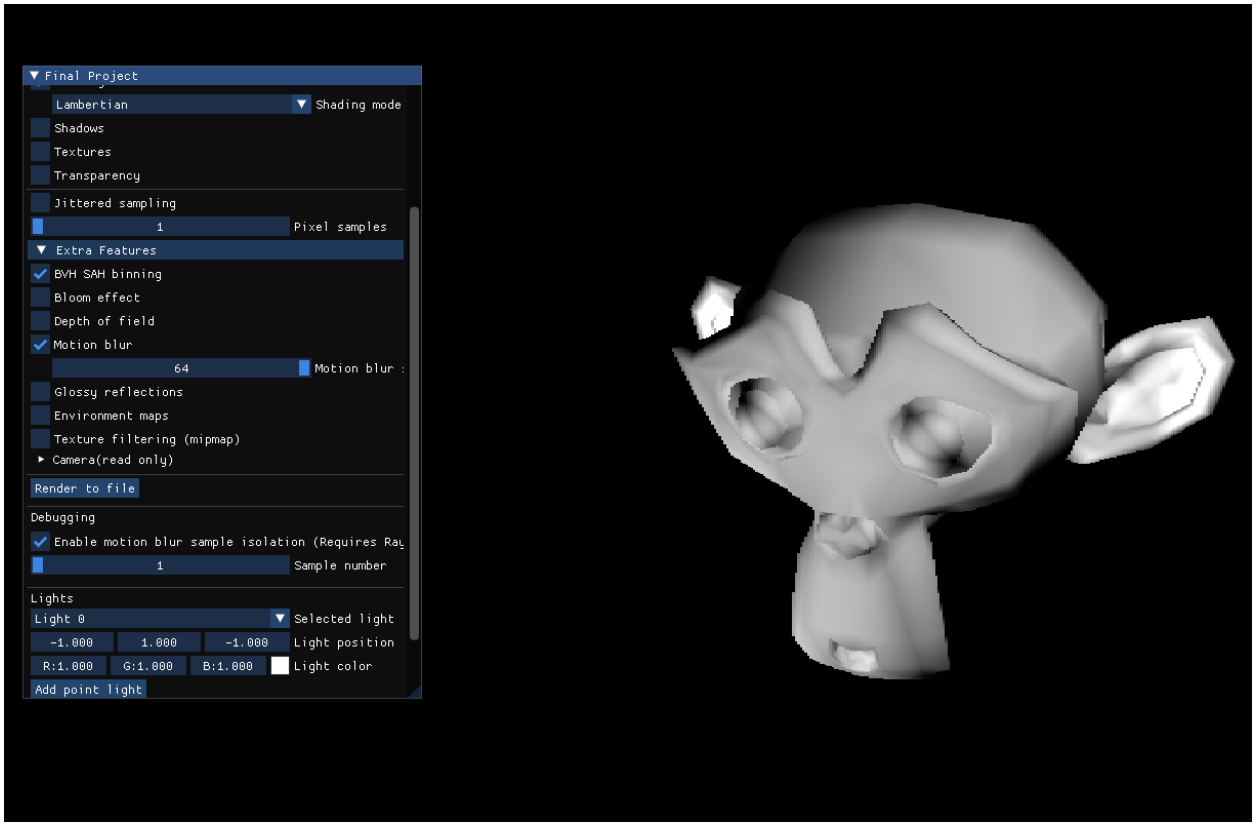
2 Samples Render



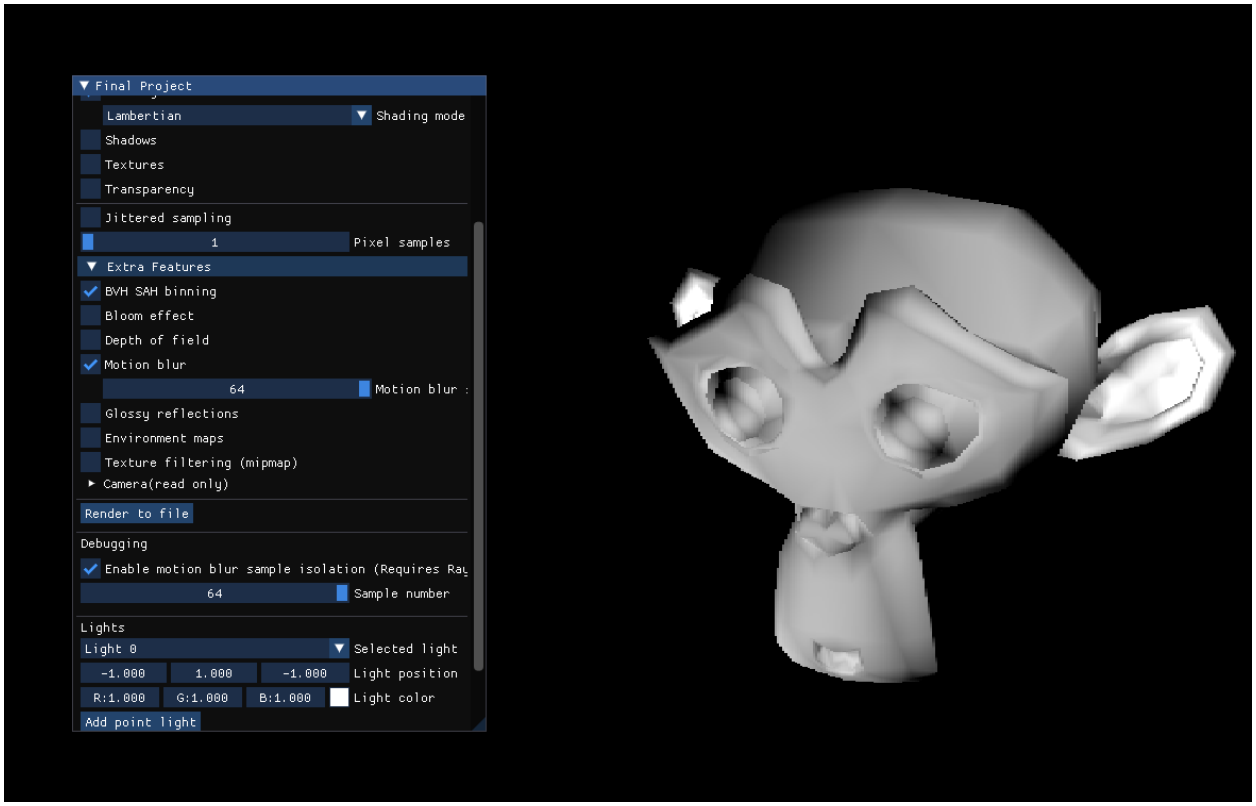
20 Samples Render



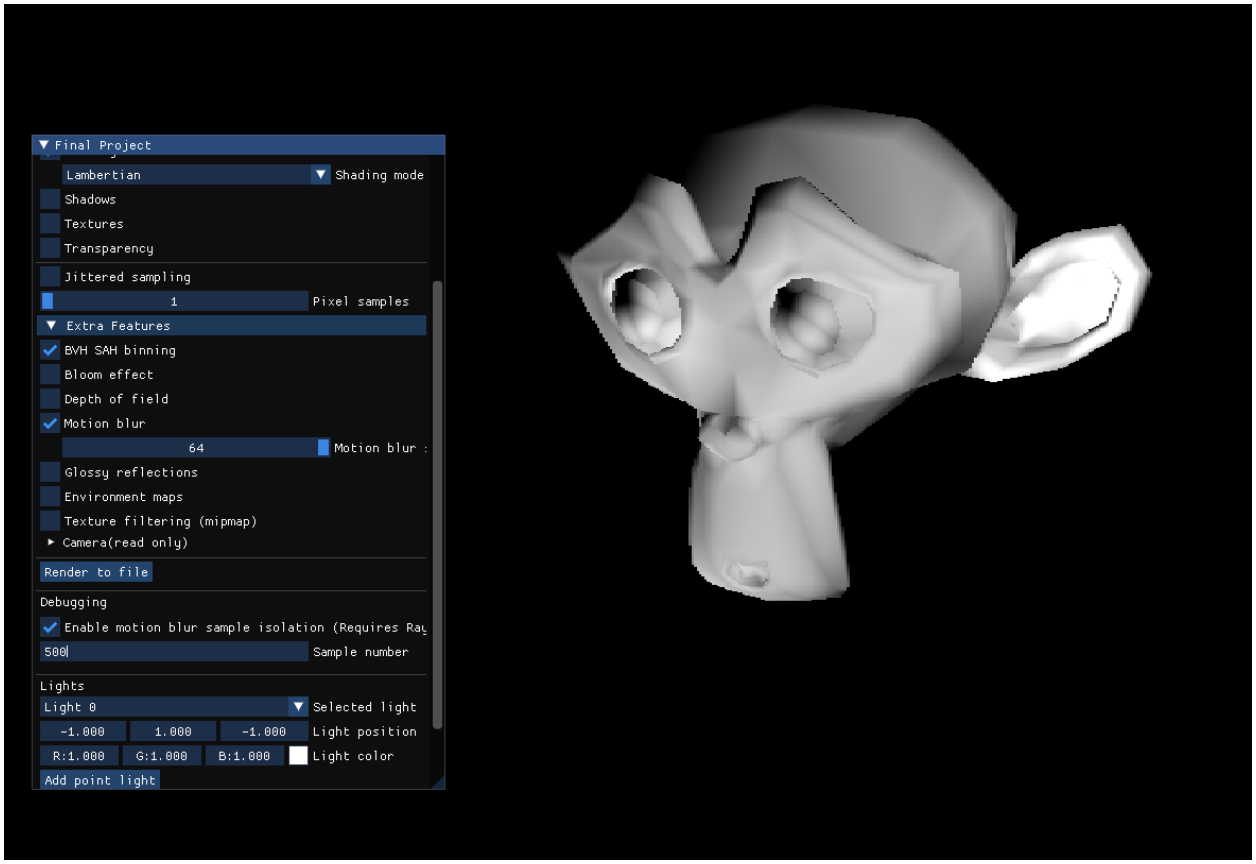
64 Samples Render



Isolated Sample #1



Isolated Sample #64



Isolated Sample #500