A5 Project Proposal
An Incremental Parallel Ray Tracer
Name: Richard Ye
Student ID: 20418361
User ID: rl2ye

Final Project:

Purpose :

To create an interactive ray tracer which supports additional material characteristics, implements parallelism to improve rendering speed, and uses incremental rendering to allow user interaction in real time or near real time.

Topics :

- Material and surface characteristics, such as refraction and glossy surfaces
- Ray tracer optimizations, including concurrency
- Adaptive anti-aliasing techniques

Statement :

This project is an extension of the work completed in A4, that is, a ray tracer with additional functionality. The implementation language has been changed to JavaScript with the modeling language changing to JSON.

The following ray tracing features will need to be implemented:

- Reflection and refraction
- Glossy surfaces
- Adaptive anti-aliasing
- Texture mapping
- Gloss mapping
- Soft Shadows
- Parallel processing
- Incremental rendering

The following UI features will need to be implemented:

- UI features to specify which features the scene should be rendered with
- UI features to modify the camera
- Optional bonus feature: UI features, including picking, used to modify the scene primitives, lighting, and other parameters.

This set of features provides an extensive challenge and substantial novelty. The first challenge is from the choice of language; JavaScript development is still in its infancy for graphics, and few ray tracers have been written in it. Furthermore, the lack of mathematical libraries means that data structures like vectors and matrices, and their associated operations, must be implemented. Since no pre-built abstractions will be

used, I hope the exercise of writing this functionality will prove useful for my understanding of writing efficient data structures at the lowest levels.

The ray tracer features selected offer a decent challenge; the algorithms behind reflection, refraction, glossy surfaces, texture mapping, etc. are well-documented but non-trivial. However, I personally find the addition of parallel processing and incremental rendering to particularly novel in addition to being both challenging and interesting.

Incremental rendering deserves a detailed explanation. It is expected that ray tracking will be too slow for real-time rendering at full resolution and quality. However, if we were to make the resolution sufficiently low, modern CPUs should be able to handle rendering in real-time. But, we would lose out on image quality. To get the best of both worlds, an incremental renderer is proposed, wherein a low-quality image is rendered in real-time upon changes to the scene parameters, with additional threads in the background progressively increasing the image quality. Both the initial render and subsequent improvements will be presented to the user in real time. Especially in combination with concurrency, this feature offers a significant challenge in the management of data and messages between threads. Thread safety, efficient data structures and memory management must be considered when implementing this feature. However, the fact that this is a challenge also offers a substantial learning opportunity for parallel processing.

The incremental renderer should also support anti-aliasing; the image should continue to increase in fidelity through the rendering of subpixels. An adaptive anti-aliasing technique should be employed to detect areas of high contrast or edges, and those areas should be anti-aliased first.

The scene to be rendered is a simple chessboard. The pieces will be modelled using standard geometric primitives and/or defined models. The pieces on one side will be transparent and will use refraction; the pieces on the other side will be opaque and will use reflection and glossy surfaces. The board itself will be a textured checkerboard with an additional gloss map to have one color of tile be shinier than the other.

Technical Outline :

The project will be written in ECMA JavaScript 1.8.5, with UI elements written using HTML elements and styled with CSS, in line with modern web development standards. Scenes will be defined using JSON files.

Program communication

Input: JSON files to define the scene. Any web standard image format (e.g. PNG) to define textures. Grayscale image to define gloss maps.

Interaction: A web page with HTML form elements to specify parameters for the ray tracer. Keyboard controls for the camera. Optionally, mouse controls for manipulating the scene.

Output: An image to screen via web page, with download link for the image.

Implementation Details

The features of reflection, refraction and glossy surfaces are well-covered in any graphics textbook, including in the textbook by Hughes, et al. [Hughes] recommended in the class. These three features are fairly straightforward to implement in a ray tracer, and I plan to follow the techniques provided therein for those features.

I shall now elaborate on what is meant by parallelism. Since each pixel (or sub-pixel) in a ray tracer is rendered independently, it is intuitive for us to take advantage of modern multi-core processors by rendering multiple pixels at once in parallel. ECMA JavaScript defines the Web Workers API which is the only way to achieve true parallelism (i.e. spawning new threads in the OS) in JavaScript. Data structures and message passing must be written to be compatible with the API. It is expected that this rendering technique will result in appreciable improvements in render time on modern multicore systems. [Horiguchi]

Thread safety must also be respected, however, luckily the language and the API was designed to inherently prevent such problems. [Moz]

The second implementation feature is incremental rendering, based on the BRL-CAD ray tracer. [Muuss] In the BRL-CAD ray tracer, the image is first rendered in a low resolution, and displayed to the user while incrementally higher resolution images are generated. I hope to combine this idea with the ideas of parallelism so the user will be able to see the image incrementally improve as time passes. Ideally, this ideal will allow real-time rendering of adequate images, with high-quality images still possible should the user choose to wait. This idea will be applied to anti-aliasing as well; should adequate time be given the renderer should start to render sub-pixels to improve overall fidelity.

The program uses a web browser and thus should use modern web development techniques. Thus the input shall be a scene consisting of a JSON file, and rendering parameters can be edited via form fields and UI elements on a web page. The output should also be a rendering area of the web page, with an option to download the resultant image. The camera should be able to be moved via the WASD keys; as mentioned earlier the image should update in real time in response to the camera movements.

The format and specification JSON file specifying the scene shall be defined in the user manual subsequent to project completion. Defining the full specification before is difficult, without knowing the anticipated details of the renderer itself. However, it should be fairly straightforward given that the SVG vector graphics format uses the competing XML format, and easily handles the definition of 2D scenes.

Adaptive anti-aliasing shall be accomplished through a simple edge detection algorithm. The initial pass shall render one ray per pixel. The pixels shall then be analyzed

to detect sharp contrasts by detecting whether a pixel differs from their neighbor too greatly. If so, anti-aliasing shall be applied.

Gloss mapping shall be accomplished by furnishing a gloss map along with a texture map. This gloss map shall be a greyscale image, with darker values representing less glossy areas of the surface.

Soft Shadows shall be implemented by defining area lights (instead of point lights) and casting multiple shadow rays per light. Area light sources shall be circles defined by a center and a radius. [Hughes]

Real-time rendering using the incremental renderer may not be possible in combination with glossy surfaces and soft shadows, due to their computationally intensive nature. In this event, there should be a UI warning notifying the user that their selected render setting precludes the use of the real-time renderer.

Deliverables :

The project will be accessible online at `http://www.yerich.net/projects/cs488/` once completed. Any documentation, manuals, data files, and source code will be clearly linked from that page and available for download.

Bibliography :

[Horiguchi]: Horiguchi, S., et al. *Parallel Processing of Incremental Ray Tracing on a Multiprocessor Workstation.* 1991 International Conference on Parallel Processing.

[Hughes]: Hughes, John F., et al. *Computer Graphics: Principles and Practice.* Addison Wesley, 2014.

[Moz]: Mozilla Corporation. *Using Web Workers.* `https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/basic_usage`. Mozilla Developer Network, 2014.

[Muuss]: Muuss, Michael John. *Towards Real-Time Ray-Tracing of Combinatorial Solid Geometric Models* . Proceedings of BRL-CAD symposium, 1995.

Objectives:

Full UserID:_____ **Student ID:**_____

___ 1: **Reflection and refraction.** Allow the definition of reflective and refractive coefficients for primitives in the scene via JSON.

___ 2: **Texture mapping.** Allow the definition of texture images for primitives in the scene via JSON.

___ 3: **Glossy surfaces.** Allow the definition of gloss coefficients for primitives in the scene via JSON.

___ 4: **Gloss mapping.** Allow the definition of greyscale gloss maps for primitives in the scene via JSON.

___ 5: **Parallel rendering.** Allow multiple rendering threads to run at the same time. Demonstrate an improvement in rendering speed.

___ 6: **Incremental rendering.** Start with a real-time render of a low-quality image, and incrementally increase the image quality.

___ 7: **Incremental rendering for anti-aliasing.** Include anti-aliasing in the incremental renderer.

___ 8: **Adaptive anti-aliasing.** Determine which pixels are in the most need of anti-aliasing using an edge detection technique. Demonstrate performance gains without noticeably sacrificing image quality.

___ 9: **Soft shadows.** Allow area light sources which cast soft shadows on objects.

___ 10: **Rendering of the final scene.** A realistic scene of a chessboard that demonstrates the accomplished objectives.