Teaching Computer Graphics During Pandemic using Observable Notebook

Sumanta N. Pattanaik and Alexis Benamira

Department of Computer Science
University of Central Florida, Orlando, USA

Abstract

Most schools and colleges around the world have resorted to remote teaching during this Covid-19 pandemic. Faced with this challenge we experimented with a novel supplement to standard video lecturing to teach our normal face-to-face large undergraduate graphics class (close to 150 students). This supplement involved using Observable notebook, and turned out to be very effective for remote teaching. We wish to continue its use even after college life turns normal, and would like to share our experience with the community and encourage such use for teaching introductory computer graphics courses.

Categories and Subject Descriptors (according to ACM CCS): K.3.2 [Computer Graphics]: Computer and Information Science Education—

1. Introduction

Teaching Computer Graphics involves teaching diverse topics such as mathematics, physics (optics), and programming, which in itself is not much different from teaching any other engineering field. However, the challenging part of teaching computer graphics is that many enrolling in a computer graphics class do not have the required preparation in basic vector and matrix algebra and optics. To make things more complicated, making a connection with the concepts in these topics to rendered visuals is not very intuitive, at least in the beginning. Furthermore, even though most have some programming background, they do not have any experience with writing relatively large programs required for even simple 3D rendering. Multiple approaches have been used by instructors to address this challenge. These approaches may be categorized as bottom-up, top-down, and hybrid approaches. For a brief description of these categories, the reader may refer to [SWL17]. For our Fundamentals of Computer Graphics class, our approach may be considered as bottom-up. It starts with teaching vector algebra and transformations and then proceeds to cover rasterization-based rendering pipeline, shader programming, fragment shade computation using light-matter interaction models, texturing, post-shader fragment operations, interactive rendering, and ends with a final project. Teaching all these concepts requires extensive use of whiteboard illustrations, interactive demonstrations, and multiple hands-on practice sessions, all of which could be well managed with face-to-face teaching and lab sessions. Since the release of the OpenGL port of JavaScript, WebGL [Khr17] has become our choice of 3D graphics programming API for interactive real-time rendering in a platform-independent way. The use of WebGL has no doubt required us to spend additional lab hours on teaching basic HTML and JavaScript. However, this additional effort is well justified because its integration with HTML Canvas and JavaScript allows the students to design interactive rendering applications without resorting to any external library download and setup effort [Ang17]. The Covid-19 pandemic has forced educational institutions worldwide to resort to remote teaching. The face-to-face classroom teaching has been replaced by video lecturing using Zoom or a similar video conferencing app. Video lecturing has allowed instructors to deliver the lectures in real-time, record their lectures and release them to the students for their reference at their own time. It has also made scheduling office hours (one-to-one meetings) relatively easier. However, such a mode of instruction has become plagued with numerous problems. If we ignore technical and technological problems, the major problems faced by instructors and students have been the lack of personal interaction and the difficulty in engaging the students remotely. These latter have been detrimental for effective teaching topics in technical areas and have been a major source of disappointments among the students. So the instructors have resorted to developing supplements to the video lecturing for alleviating the problem. We describe here our supplemental approach for remote teaching of our undergraduate level fundamentals of Computer Graphics class.

2. Methodology

Two main ingredients of our approach are: (i) Use of Observable notebook for interactive illustrations and algorithm demonstrations, (ii) use of REGL-WebGL framework on Observable notebook for graphics programming. We explain below the reason behind such uses.

DOI: 10.2312/eged.20211007

2.1. Observable Notebook

Observable notebook is a computational notebook that shows as a document on browsers [A*19]. It is a new addition to other existing notebooks such as Jupyter [PG], RNotebook [McP16] that are receiving acceptance as excellent tools for teaching. Observable notebook shares the features of other notebooks in that a notebook page is composed of computation cells, but differs in most other aspects. Observable notebook pages are composed of a single type of cell, code cell. Each cell holds a single JavaScript statement or a block of statements. Every cell returns a value. The cell containing a block of statements must contain a return statement to return a value computed in the code block, otherwise it returns *undefined*. The cell may define a function as well. The cells are executed by the client browser first when the page is opened. The output of the cell appears above the cell. The cell code that produces it appears below. The code can be made visible or invisible by the viewer. The output of a cell can be any rendered HTML element: image, illustration in Canvas or as SVG element, formatted text; an interaction widget; a number; a string; a function; or a JavaScript object. A cell can be added to the opened notebook by the notebook viewer. Any cell appearing on the notebook can be edited by the viewer as well. The cell code is executed automatically once it is created or edited. The cell results can be named and accessed as constants by other cells on the notebook page. Unlike other computational notebooks in which code in notebooks is typically executed interactively, one cell (or block of cells) at a time, Observable notebook cells are reactive. Observable notebook cell names can be accessed as constants, in any order and the change in their values trigger the re-execution of all the cells that use the cell names. The JavaScript code in the cell uses an extended version of JavaScript that has a built-in library to create HTML input and output elements. For example: it can use *DOM* library object to create HTML and SVG elements; it can use independent functions such as *html* to create HTML element, *md* (for Markdown) to create formatted HTML paragraph, *svg* to render SVG, and *tex* to render Latex. Using *require* the notebook allows the import of JavaScript libraries that are published through NPM and using *import* it allows the import of code and data from other published notebooks into the current notebook.

Observable notebook is designed for rendering data. Rendered data, maybe an SVG plot, an HTML table, a Context2D or WebGL rendering on a Canvas. 2D libraries such as D3 [Bos20] and 3D libraries such as ThreeJS [(Mr20] are being widely used for rendering data on web pages. Such data rendering capabilities make the Observable notebook ideal for creating static and interactive illustrations and renderings along with written explanations using HTML and Markdown and Latex formatting capabilities. JavaScript Code broken down into small code snippets in independent cells makes it easier for the viewer to navigate, understand and debug complex rendering concepts. Independent editing capabilities allow the viewer to modify the data/code to see their immediate effects on the rendered visual elements and thus facilitate learning. The concepts of interactive Code editing and viewing are not new. Tools like JSFiddle [KZ20], CodePen [Coy20] allow the user to edit the code and see the effect results side by side on the browser. However, the fine granularity provided in the form of cells and the capability to show the result of independent execution of the cell makes the Observable notebook a much better tool for learning.

2.2. REGL: A WebGL Wrapper

WebGL is the JavaScript binding of OpenGL ES API. WebGL conforms to OpenGL ES2 and ES3 standards. It is designed to run on browsers to carry out GPU accelerated 2D/3D rendering on HTML Canvas. WebGL is currently natively supported by all browsers. Because of the OpenGL ES compatibility and browser support, a WebGL-powered HTML page runs on any smart platform. A WebGL program has a control code component that is written in JavaScript to prepare the data for GPU pipeline and to interface with GPU driver to pass data and control commands. It also has a shader code component that is written in GLSL to carry out actual rendering in GPU. Hence WebGL programming is time-consuming and has a steep learning curve. A number of utility libraries have been developed [Seg] to encapsulate both these coding parts into simpler function calls to make WebGL-driven rendering application development easier. However, full-encapsulation, as is done in ThreeJS, takes developers farther away from the fundamental rendering concepts and WebGL programming, and focuses more on the application development. So we decided to use a utility library that encapsulated only the verbose coding part (control coding). Of the available choices we chose REGL, a WebGL wrapper [L*], that provides an object orietned abstraction to simplify the command part into an object. From this abstraction, it internally creates a function that includes all the required WebGL states and draw calls, which can be used for rendering. The user must write the full shader code to carry out any rendering. Figure 1 lists a typical code template for mesh drawing using REGL.

```
const drawMesh = regl({
  frag: `...`,
  // Multiline Fragment shader code in GLSL
  vert: `...`,
  // Multiline Vertex Shader code in GLSL
  attributes: {
    <attribute name> : <data array>,
    // Prepares data buffer and connects
    // shader attribute to data buffer.
    ...
  },
  uniforms: {
    <uniform variable name> : <value>
    // Sets the shader uniforms with value.
    // Dynamic setting at draw time is possible.
  },
  count: <number of vertices in the mesh>
});

regl.clear({color: [...]}) // Clear Canvas

drawMesh();
// WebGL states and draw calls are instantiated
// to initiate shader execution and rendering.
```

Figure 1: *Sample REGL renderable template.*

The object-oriented abstraction removes the overwhelming details of the control programming and focused mostly on the shader

programming and data directly, and hence makd it easier to adapt for my class students most of whom had sufficient OO Programming background. REGL's low footprint on Observable notebook collections also made it an attractive choice for the class, because it enforced learning by forcing the students to go over the sample notebook pages supplied in the class to complete their assignment, instead of passing on an existing notebook page as their own work.

2.3. Notebook Pages as Teaching Supplement

We created several interactive illustrations and practice WebGL notebooks. Most of the illustrations were created using D3 library (a 2D drawing API) or using D3 based functions imported from other published notebooks. The illustration notebooks were designed to enforce learning of basic concepts, such as vector algebra, transformations, the functioning of reflection models. Practice WebGL notebooks were created to teach WebGL programming and some of the basic shading concepts. WebGL notebooks used REGL, and hence contained mostly data, shader code and interaction widgets, and the rendered output. The students were required to extend them as a part of their weekly assignments. All the notebooks were used for interactive demonstrations during Video lecturing. The illustration and practice WebGL notebooks were released to the class before the class hour so that the students could use them during the class, interact and modify them as the concepts were covered, and ask for clarification if required. Hence, they served as practice lessons. Weekly assignments were also demonstrated using the notebook and discussed in the class but were released only after the assignments were due to let those who had difficulty completing a particular assignment catch up with the class and continue with subsequent assignments.

We list here the main categories of notebook pages that we created for the course. All the notebooks listed here and many more are published, and are freely available as a collection from https://observablehq.com/collection/@spattana/class-4720 [Pat20a].

- Vector Algebra and Transformations: We created interactive pages to illustrate the concept of coordinates and vector operations. Figures 2, 3, 4 show the screenshots from http://bit.ly/2PRsEqE and http://bit.ly/2OlDVPQ that illustrate vectors and vector operations in 2D. Figure 5 shows the screenshots from http:http://bit.ly/30CoTaV and http://bit.ly/3rJPjmJ that illustrate 3D rotation transformations. Rotation transformations were covered after we covered basic WebGL rendering of 3D objects. So, the illustrations were associated with 3D rendering and the REGL-WebGL code. Though transformation matrices and their computations were covered in the lectures, we encouraged the students to use functions from glMatrix library [J*] to avoid any unforeseen coding error introduced by the students while computing them for assignments.
- WebGL graphics rendering pipeline and WebGL rendering: These pages illustrate the basic functioning of WebGL rendering pipeline and basic WebGL rendering. Figure 6 shows screenshots from http://bit.ly/3bIfAwj and http://bit.ly/3rECme5 that illustrate the mathematical concept of barycentric coordinates along with its application in the rasterization and interpolation of the varying variable along the pipeline and introduce the concept of aliasing. Figure 7 from http://bit.ly/38yCiF2 introduces the basic WebGL rendering steps starting with clearing the canvas and changing uniforms for interactive changes in the rendered view.
- Light and Reflection Models and Shade Computation: We created a number of illustrations to introduce light types and light-matter interactions. Figures 8, 9, 10 show the screen-shots of the 2D illustrations of the concepts from http://bit.ly/3lbdvfD, http://bit.ly/3bGnwxZ, and http://bit.ly/3bGxTC8. We

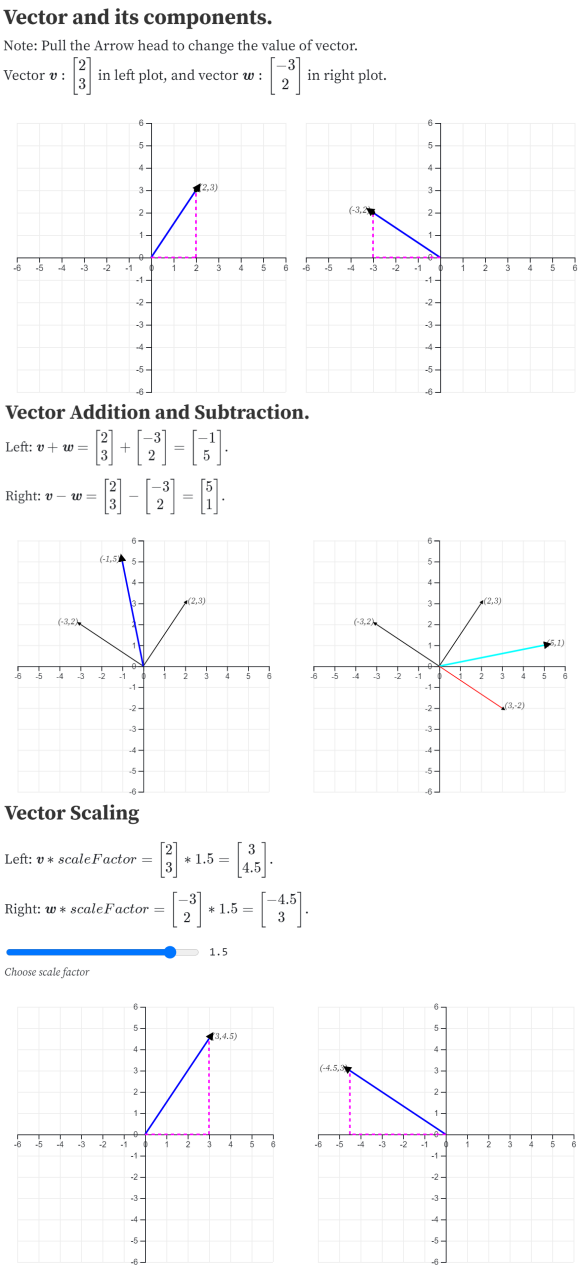


Figure 2: *Vector Operations.*

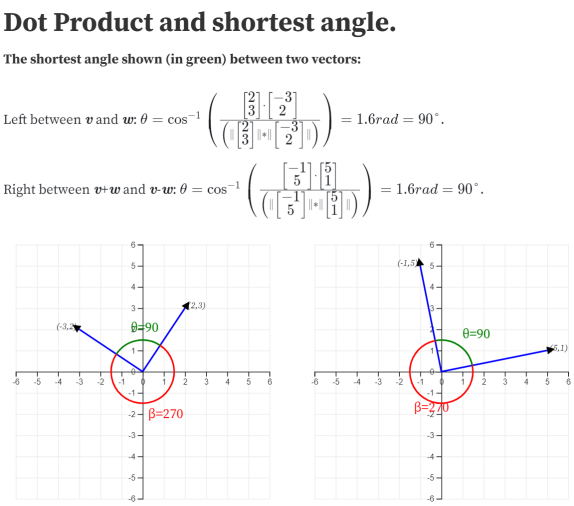


Figure 3: *Dot product and its relation with the angle between two vectors.*

also created notebooks to demonstrate shade computation in the fragment shader. However, instead of sharing them immediately we supplied a basic WebGL rendering with camera support and tasked the students to code all the lighting concepts as consecutive weekly assignments.

- Texture Mapping: We created and shared a texture rendering notebook (see Figure 11 from http://bit.ly/3ldZhum) for material property and surface roughness variation, and a notebook for skybox rendering using cubemap textures (see Figure 12 from http://bit.ly/38AQpcW). Students were tasked to use them for their rendering assignments with complex 3D models, and for environment mapping.
- Post-Fragment Shader operations: We created and shared notebooks to illustrate the use of stencil buffer (Figure 14) and depth buffer, color buffer, and blending (see http://bit.ly/2NhoI1C), and their use for mirror reflection and shadow projection rendering (Figure 13) (see http://bit.ly/3cuiJyW). These latter ones served as the last weekly assignment of a total of ten weekly assignments in which the students were asked to choose any one of the two concepts (mirror reflection or shadow projection) for rendering.
- Miscellaneous: In addition to those mentioned above, we created a number of other notebook pages that served as illustrations for demonstrations during the lectures. The screenshots of some of those illustrations are shown in Figure 15 that were used during the lectures on clipping and shadow volume algorithm (See http://bit.ly/3qGqabn, http://bit.ly/3cqAgIh and http://bit.ly/3vltMDd); in Figure 16 (see http://bit.ly/38DhmwV and similar notebook pages) on color models that were used in two full lectures devoted to discussing color concepts. A few more pages that described the concepts of parametric curves and surfaces, and of Scenegraph, animation, and instancing are not shown in this paper. All the notebooks are available as a collection from [Pat20a].

3. Discussion

The supplements discussed in the previous section may look like pages of another Web-based tutorial. There are several tutorial Web pages out there that illustrate fundamental concepts and also provide an interactive experience. Those tutorials may be considered as online replacements or supplements of textbooks with interaction as the added component. Observable notebook-based supplements that we advocate in this paper may be considered as the online replacement of practice workbooks. The cells in any notebook can be modified to instantly see the changes. The cell results appear above the cell and can be inspected without resorting to any special effort, such as writing additional debug print statements (console.log on JavaScript) and looking around in the console to find the results. A local copy of any notebook can be *fork*ed to experiment with new data, with extensions of the code, and can be saved for future use. Though it is not impossible to do the latter with existing Web-based tutorials, it is not as simple as what Observable notebooks make. The notebooks once *published* (a single click) become available to anyone with browser access on any platform. The developer of the notebook does not have to create his/her own server to host the notebooks. The *import* facility allows an instructor to create his/her own modifications to match with his/her own style of teaching. Thus, Observable notebook-based teaching helps in creating, updating, sharing, and reusing teaching resources easier.

Every class has a distribution of students, some of whom progress on their own, and some need extra help to make progress. That is why the instructor sets aside office hours to meet and help those students. However, for various reasons only a few students ask and get that help. Even when the student meets the instructor, it is a little time-consuming to look over the student's shoulder to find out where the student's code may not be performing correctly on his/her computer. Transferring the JavaScript-HTML page to the instructor's computer for debugging is even more time-consuming and frustrating. Observable notebook made this task much easier. Exactly like publishing, an Observable notebook can be made sharable instantly with a single click and the link can be shared with the instructor. The cell-wise break-up of the code and availability of cell results for viewing helps in locating the errors in the code block faster. We made use of this facility throughout the semester. We are pleased to say we were able to meet one-to-one (remotely) and help a much larger fraction of the students than we were ever able to during our normal semester teachings. The students also felt very satisfied. From our informal feedback session at the end, we gathered that the students believed they learned better from this mode of teaching and interaction.

The advantages of the Observable notebook that we mentioned in the previous paragraph also made the grading of our large class easier. Our course had weekly programming assignments. For each assignment, we either supplied a template notebook page that the student *fork*ed, or required that the students extended their previous week's assignment to complete the current week's assignment. As publishing a notebook makes it available to any web user, to avoid plagiarisation we required that the students made their notebook link sharable only (not publish), and submit the notebook link as a part of the submission. The observable notebook allows

resource access through direct URL fetch and through file attachments, where files were stored in the Observable server cloud. So we did not face any problem of missing data or resource files during the assignment evaluation. As for the file attachment, there is no restriction on the type of file, but there is a size restriction of 15MB per file. The size restriction did not cause any problem for the assignments.

For final group projects, the students from each group were tasked to work, complete, and submit their work using a single Observable notebook. Observable allows group collaboration on a notebook using its real-time multiplayer editing and commenting facility [B*]. However, this latter feature requires a feepaying membership. Though membership cost is small for the duration of the project (about a month) we encouraged the students to collaborate through *fork*ing and merging. Of a total of fifteen projects (with an average of about ten students per project) to our knowledge, only one group resorted to a fee-paying multiplayer editing facility, whereas the rest worked on the project using *fork* and *merge*. None of the groups complained of any inconvenience in such use. Only one of the projects (large point cloud viewing) had to reduce their data size because of the size restriction of the local file attachment. The class projects are all published and are available as an Observable notebook collection [Pat20b].

4. Summary

We used Observable notebook for creating and sharing supplementary teaching material for effective remote teaching of fundamentals to Computer Graphics. Though we have not carried out any formal study to measure the effectiveness, we gathered from the end of the class informal feedback session that students were satisfied with the shared supplementary material, the improved quality of teaching, and one-to-one attention. They felt that similar use of Observable notebooks in other courses would be helpful. Observable notebook use certainly helped us in teaching and grading.

We created close to 30 notebooks for the class during the course of the semester, most of which were developed during the Fall semester itself. We have added more since then. We plan to continue using the created materials even after the semester teaching gets back to normal. We do not claim that the created course supplements are sufficient but would continue adding newer notebook pages to fill in the gaps. The created materials are all published in an Observable notebook collection [Pat20a] and hence are freely available to anyone interested in using them. We encourage instructors to use them, create more and share with others in the community.

References

[A*19] ASHKENAS J., ET AL.: *Observable User Manual*, 2019. URL: https://observablehq.com/@observablehq/user-manual. 2

[Ang17] ANGEL E.: The case for teaching computer graphics with webgl: A 25-year perspective. *IEEE CG&A March/April* (2017), 106–112. 1

[B*] BOSTOCK M., ET AL.: *Observable Teams*. URL: https://observablehq.com/teams. 5

[Bos20] BOSTOCK M.: *D3: Data Driven Documents*, 2020. URL: https://d3js.org/. 2

[Coy20] COYIER C.: *CodePen: Online Code Editor and Front End Web Developer*, 2020. URL: https://codepen.io/. 2

[J*] JONES B., ET AL.: *glMatrix.js*. URL: http://glmatrix.net/. 3

[Khr17] KHRONOS W. W. G.: *WebGL2: Web 3D Graphics Library*, 2017. URL: https://www.khronos.org/webgl/. 1

[KZ20] KRAWCZYK O., ZALEWA P.: *JSFiddle: An online IDE*, 2020. URL: https://jsfiddle.net/. 2

[L*] LYSENKO M., ET AL.: *Regl Project*. URL: https://github.com/regl-project/regl. 2

[McP16] MCPHERSON J.: *R Notebooks*, 2016. URL: https://blog.rstudio.com/2016/10/05/r-notebooks/. 2

[(Mr20] (MR.DOOB) R. C.: *Three.js – JavaScript 3D library*, 2020. URL: https://threejs.org/. 2

[Pat20a] PATTANAIK S.: *Computer Graphics Fundamentals*, 2020. URL: https://observablehq.com/collection/@spattana/class-4720. 3, 4, 5

[Pat20b] PATTANAIK S.: *Fall Computer Graphics Fundamentals Class project Collection*, 2020. URL: https://observablehq.com/collection/@spattana/cap-4720-2020-class-projects. 5

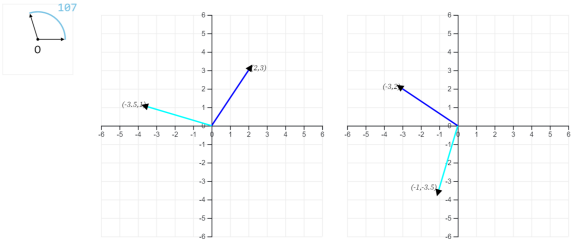
[PG] PÉREZ F., GRANGER B.: *Project Jupyter*. URL: https://jupyter.org/. 2

[Seg] SEGUIN D.: *WebGL WebGPU frameworks libraries*. URL: https://gist.github.com/dmnsgn/76878ba6903cf15789b712464875cfdc. 2

[SWL17] SUSELO T., WÜNSCHE B., LUXTON A.: The journey to improve teaching computer graphics: A systematic review. In *International Conference on Computers in Education* (2017). (Proc. Eurographics'17). 1

Rotation Matrix

Rotation by Angle 107°: Left: Rotation of vector $\begin{bmatrix}2\\3\end{bmatrix}$, Right: Rotation of vector $\begin{bmatrix}-3\\2\end{bmatrix}$



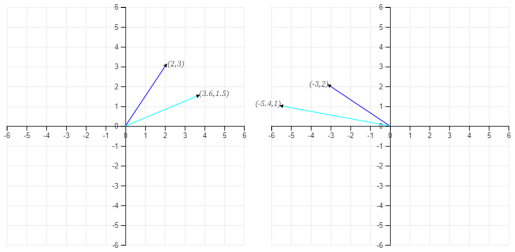
$$\text{Rotate}(\boldsymbol{v}) = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} * \boldsymbol{v} = \begin{bmatrix} -0.296 & -0.955 \\ 0.955 & -0.296 \end{bmatrix} * \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} -3.5 \\ 1 \end{bmatrix}.$$

$$\text{Rotate}(\boldsymbol{w}) = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} * \boldsymbol{w} = \begin{bmatrix} -0.296 & -0.955 \\ 0.955 & -0.296 \end{bmatrix} * \begin{bmatrix} -3 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ -3.5 \end{bmatrix}.$$

Scale Matrix

Nonuniform scaling applied to: Left: vector $\begin{bmatrix}2\\3\end{bmatrix}$, Right: vector $\begin{bmatrix}-3\\2\end{bmatrix}$

xFactor 1.8 yFactor 0.5



$$\text{Scale}(\boldsymbol{v}) = \begin{bmatrix} xFactor & 0 \\ 0 & yFactor \end{bmatrix} * \boldsymbol{v} = \begin{bmatrix} 1.8 & 0 \\ 0 & 0.5 \end{bmatrix} * \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 3.6 \\ 1.5 \end{bmatrix}.$$

$$\text{Scale}(\boldsymbol{w}) = \begin{bmatrix} xFactor & 0 \\ 0 & yFactor \end{bmatrix} * \boldsymbol{w} = \begin{bmatrix} 1.8 & 0 \\ 0 & 0.5 \end{bmatrix} * \begin{bmatrix} -3 \\ 2 \end{bmatrix} = \begin{bmatrix} -5.4 \\ 1 \end{bmatrix}.$$

Normal Matrix is Inverse Transpose of Model Matrix!

A simple non-uniform scaling transfomation $\begin{bmatrix}1 & 0\\0 & 1.8\end{bmatrix}$ is applied to the quadrilateral shown in the plot below. The vector $\begin{bmatrix}0.5\\0.9\end{bmatrix}$ drawn in red is the normal vector $\begin{bmatrix}0.7\\0.7\end{bmatrix}$ transformed using Model Matrix, and the vector $\begin{bmatrix}0.9\\0.5\end{bmatrix}$ in blue is the normal vector correctly transformed using Inverse Transpose of the model Matrix $\begin{bmatrix}1 & 0\\0 & 0.556\end{bmatrix}$ so that it remains perpendicular to the edge of the quadrilateral.

1 *Change X Scale* 1.8 *Change Y Scale*

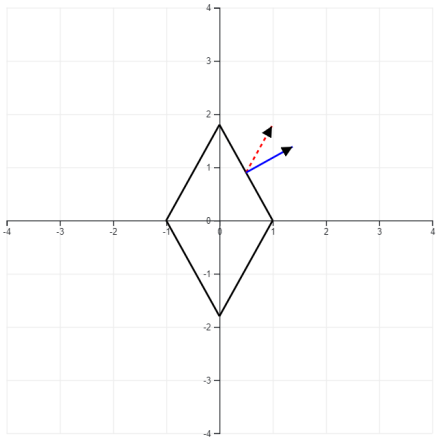


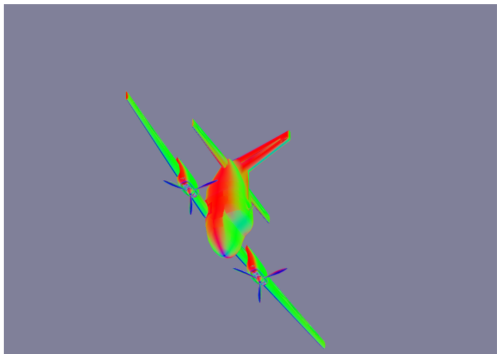
Figure 4: *2D Transformation Matrix and Transformation Matrix for normal vector.*

Rotations

Aircraft Principal Rotations:

An aircraft in flight is free to rotate in three dimensions:
yaw: nose left or right about an axis running up and down;
pitch: nose up or down about an axis running from wing to wing; and
roll: rotation about an axis running from nose to tail.
See Wiki.

Yaw 0 *Nose Left/Right: Rotate along Y-axis*
Pitch -20 *Nose Up/Down: Rotate along X-axis*
Roll -60 *Rotate along Z-axis*



Rotor Quaternion

Rotor quaternion $q = (sin(\theta/2) * axis, cos(\theta/2))$ represents rotation around a normalized vector "axis" by angle "θ".
A vector v can be rotated using a rotor quaternion using the following quaternion product:
$q * v * conj(q)$
This product may be converted to a 3x3 rotation matrix for its use in WebGL pipeline. As an alternative to computing a rotation matrix followed by matrix multiplication, the following relation may also be used to directly transform the vector using the rotor quaternion: (assumes a vec4 representation of quaternion)
$q * v * conj(q) = v + 2.0 * cross(q.xyz, cross(q.xyz, v) + q.w * v)$

See the placement of rotated discs around a unit sphere using rotor quaternion.
A disc of radius 0.1 on XY plane and centerd at origin, is oriented and placed at different locations on the surface of a unit sphere.

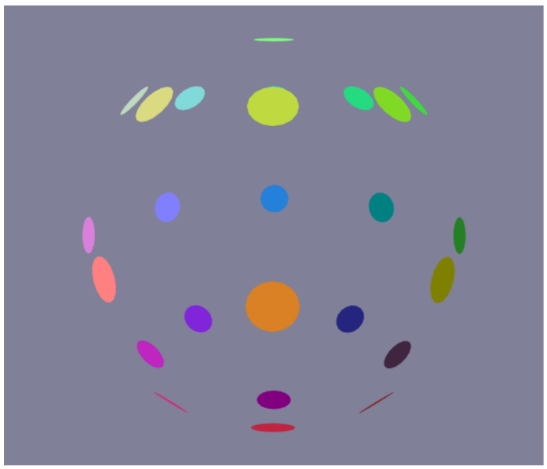


Figure 5: *3D Rotations and Quaternion for Rotation.*

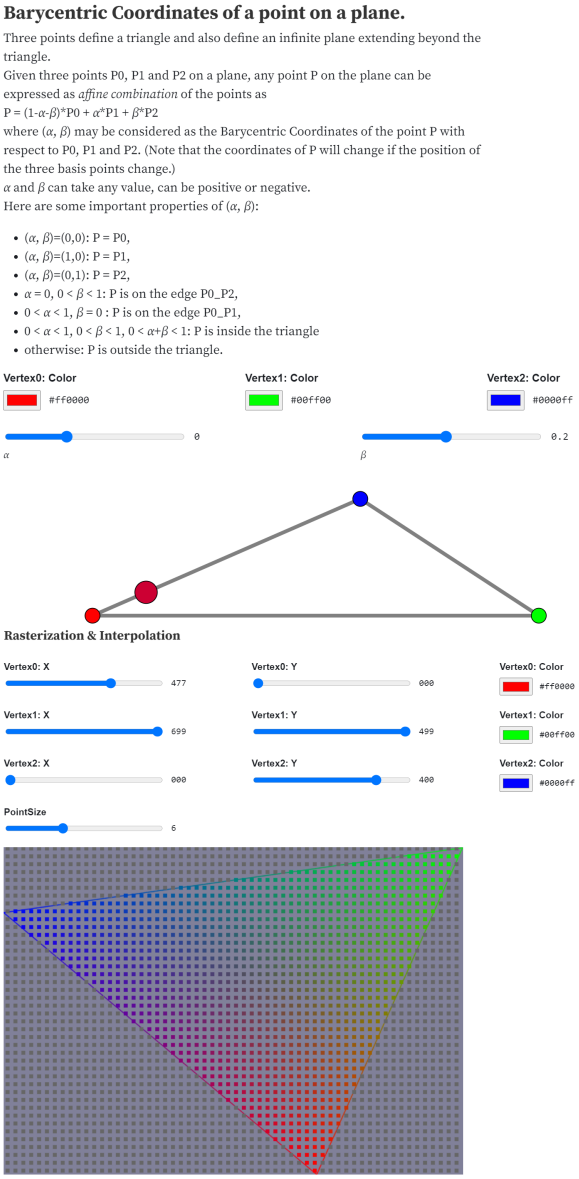


Figure 6: *Barycentric Coordinates, Rasterization and Interpolation.*

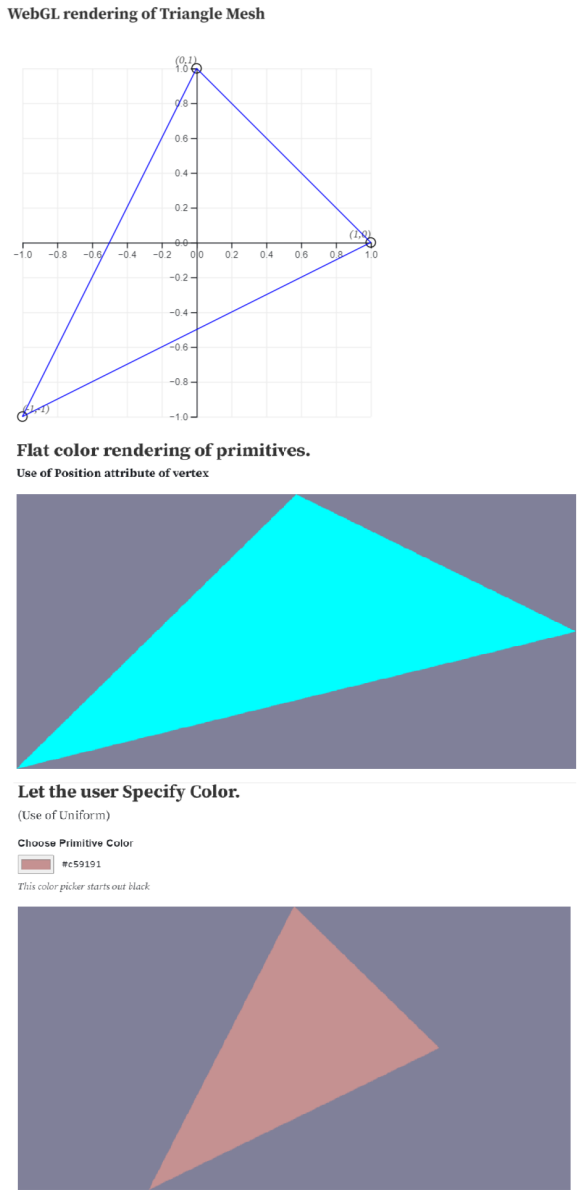


Figure 7: *Basic WebGL rendering steps.*

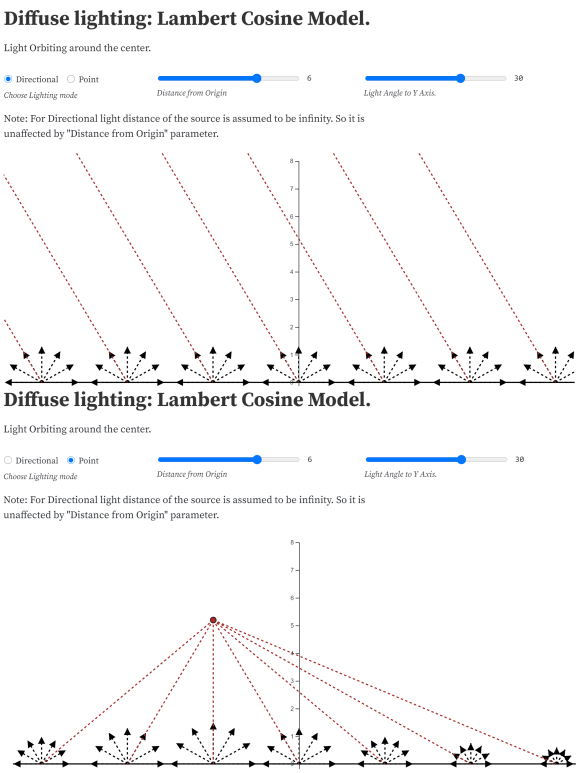


Figure 8: *Light Models and Diffuse Reflection.*

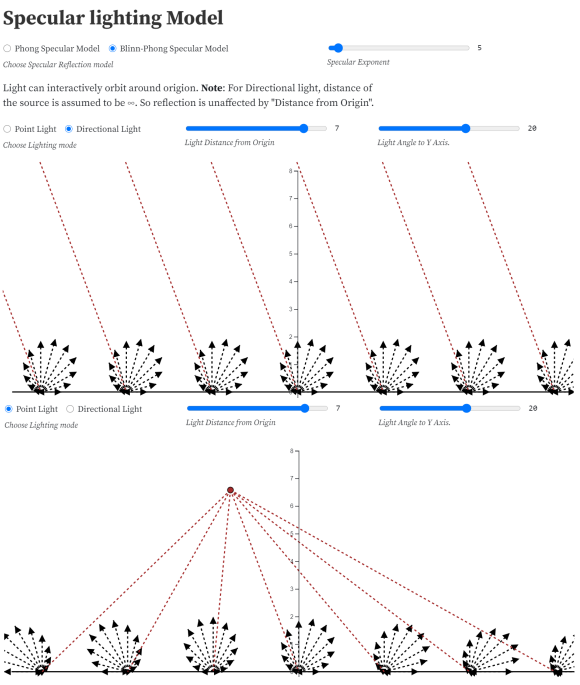


Figure 9: *Light Models and Specular Reflection.*

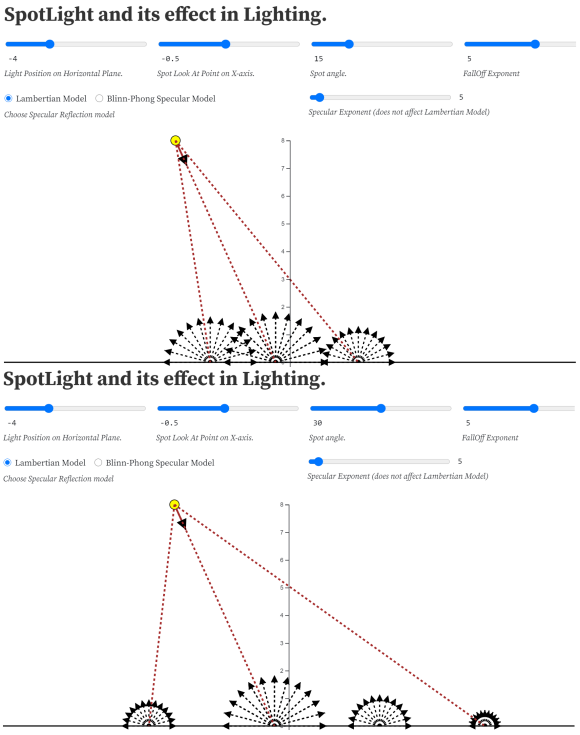


Figure 10: *Spot light.*

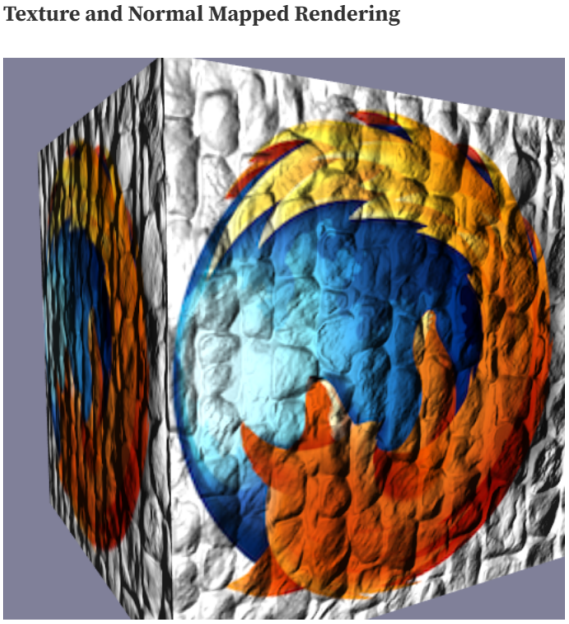


Figure 11: *Texture and Normal mapped Rendering.*

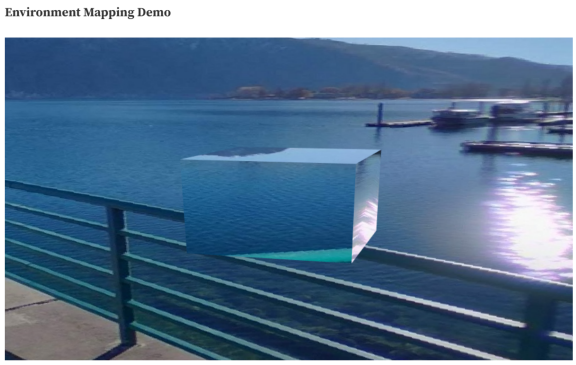


Figure 12: *Environment Mapping.*

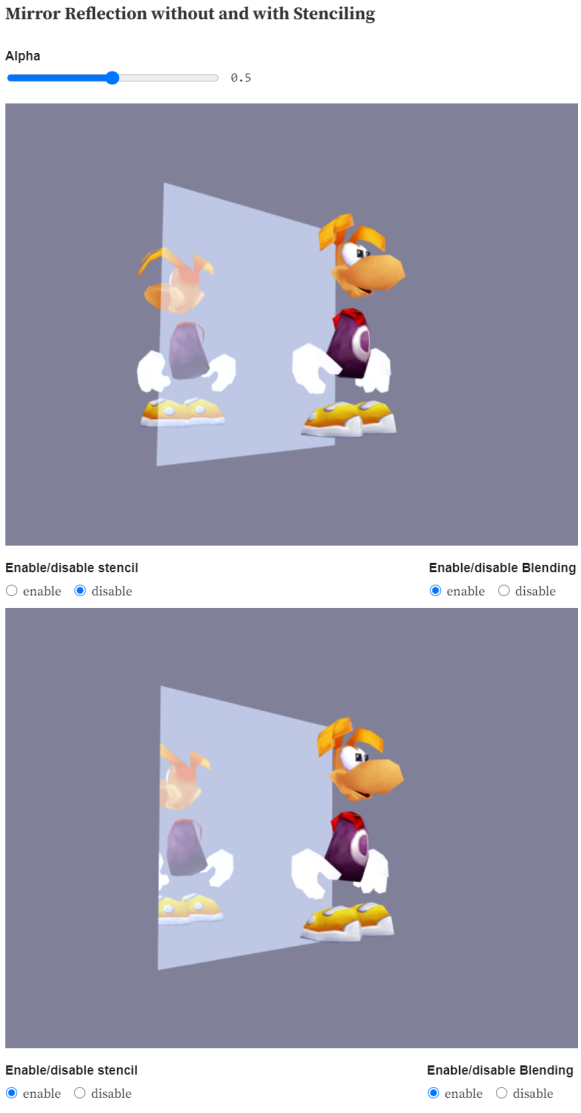


Figure 13: *Application of Stenciling and Blending.*

Figure 14: *Illustration of Stenciling Concept.*

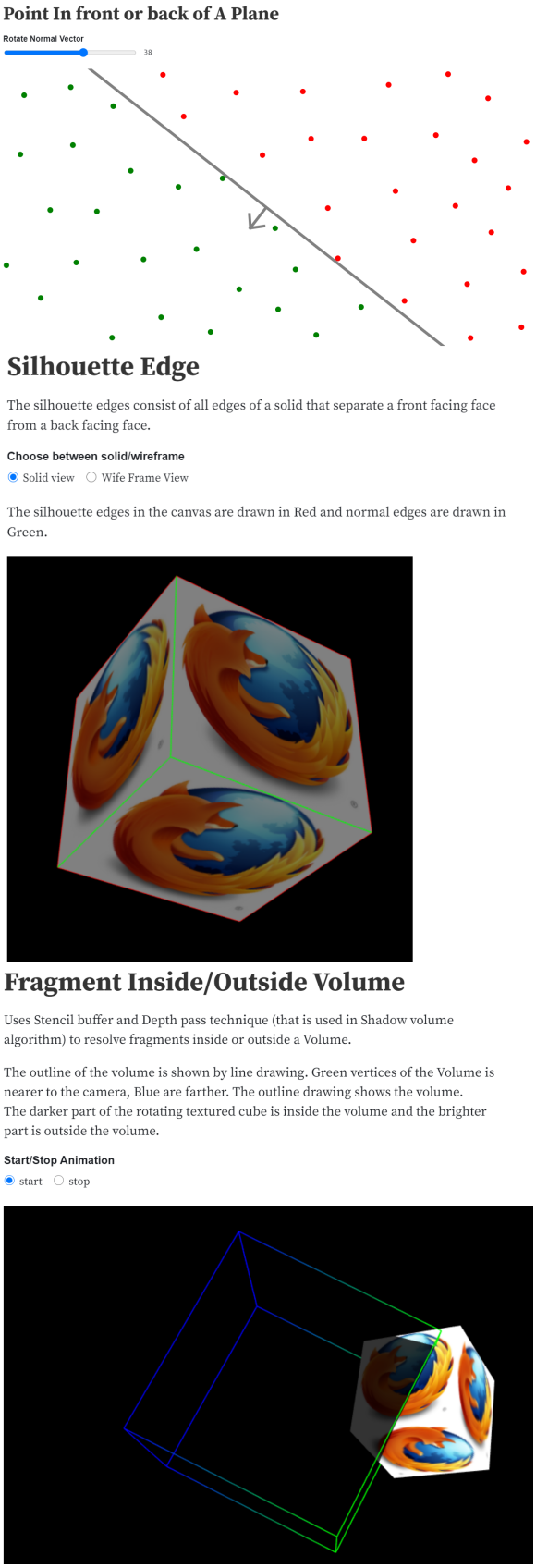


Figure 15: *Computational Geometry Principles and their applications.*

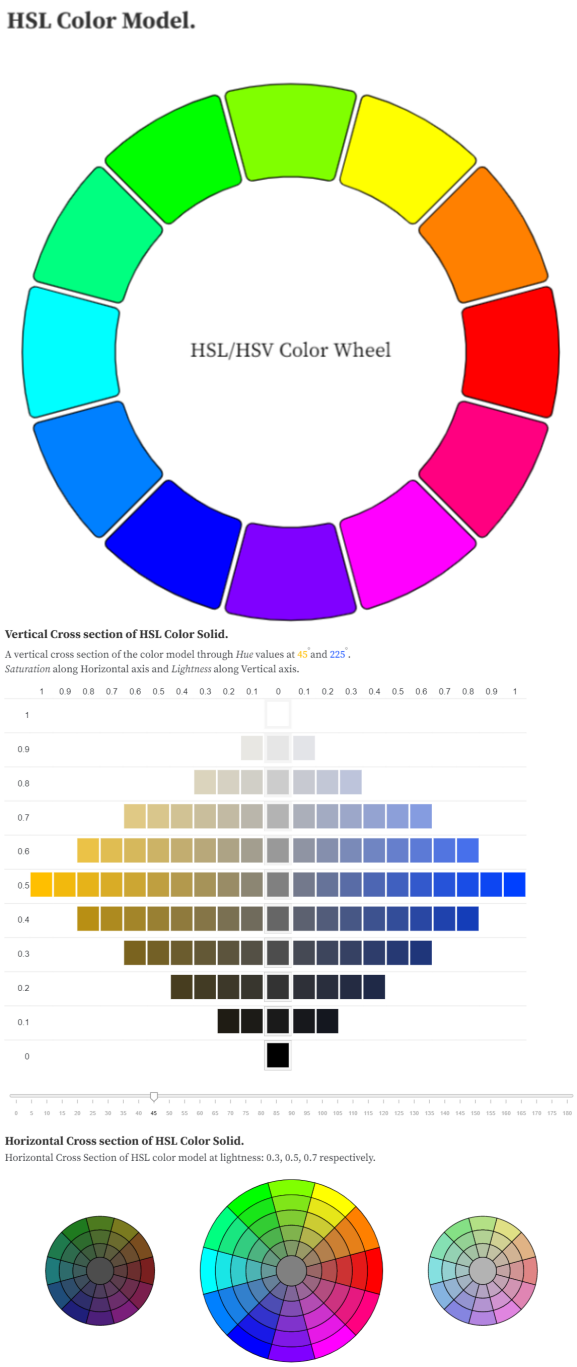


Figure 16: *HSL Color Model.*