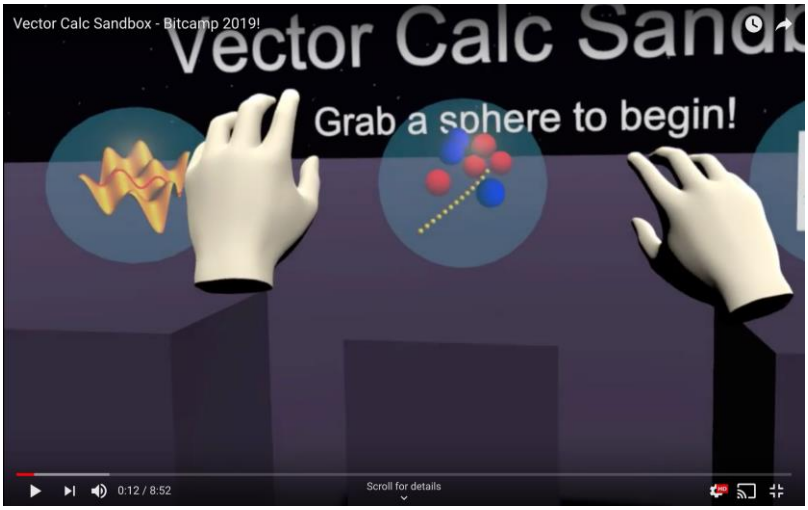
Vector Calculus Sandbox

JOY DEEP SAHA

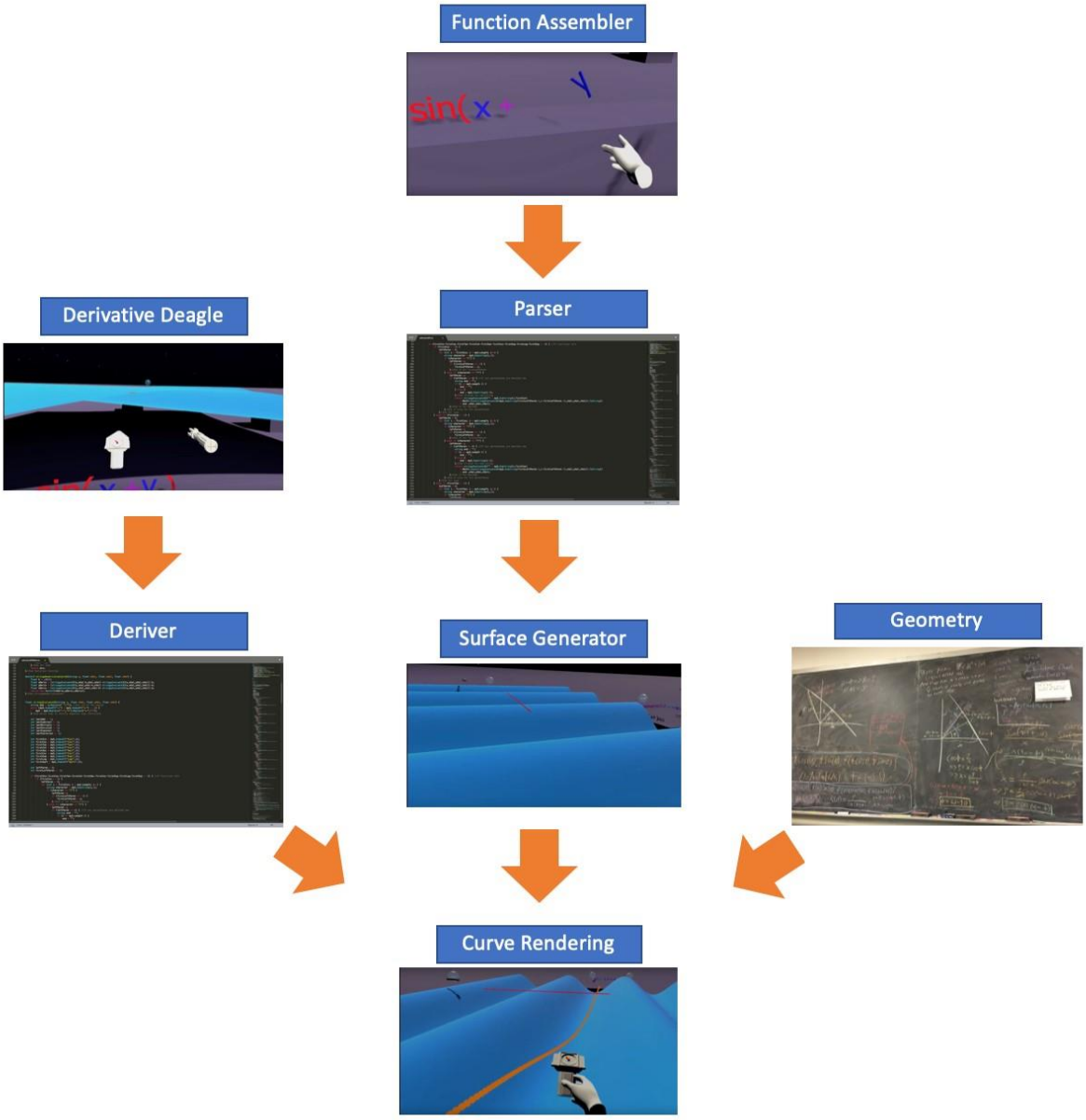
Project Description



Vector Calculus Sandbox is a virtual reality based multivariate calculus simulator, designed to be a pedagogical tool. At UMD the multivariate calculus class uses MATLAB, which many students find ineffective, so our goal was to create a more intuitive software to explain these concepts to a new student. We used the game engine Unity which utilizes scripts in C#, as well as Mathematica to confirm calculations and simulate our ideas before implementing them in Unity, although there is no part of the project that needs Mathematica to run. There are three models that the user can choose from: a directional derivative visualizer, an electromagnetism simulation, and a vector field simulation. The user wears an Oculus Rift virtual reality headset, immersing them in virtual reality, along with a controller for each hand so that they may grab things and see models of hands in front of them inside the simulation.

Directional Derivative Visualizer

The user starts by assembling a function of two variables using an interactive function builder. The user is surrounded by tools for creating polynomial functions, with the addition of trigonometric functions, the natural logarithm, and some constants. The user is surrounded by these different pieces floating in the air, and in virtual reality may reach out and grab them to be placed on a table. The pieces automatically align themselves on the table to form the function. Any well-formed function of two variables with the tools provided is useable. When the function is finished, the corresponding surface appears in front of the user, that is, a function of two variables is represented as a surface floating in 3-space. The user has two tools at their disposal – what we have called the Gradient Gun and the Derivative Deagle. The Gradient Gun can be grabbed and held in virtual reality. The user can aim the Gradient Gun at the surface to project a line to the surface, rendering a horizontal line whose length and direction correspond to the gradient of the surface at that point. The other tool, the Derivative Deagle, can be aimed at the surface to render a curve that shows the path that a directional derivative, in the direction specified by the user using the joystick on the controller, would follow. The intuition we would like to highlight is that a directional derivative is really a 1-dimensional derivative, like in Calculus I and II, so we highlight the cross section of the surface along which one can think of the directional derivative as being a 1-dimensional derivative. This demystifies the directional derivative by breaking it down into an application of a basic skill from Calculus I. The flowchart below shows the pieces and parts necessary to build this part of the project.



Function Assembler: The blocks for assembling the function were designed in a 3D modeling software and exported to be used in Unity as .obj files. A large amount of front-end work was done to make a smooth interface to grab the symbols and have them snap onto the table into an expression.

Parser: Coding this was a challenge as Unity uses C#, neither of which have built in standard calculus or even equation evaluating packages, meaning that we had to create a parser from scratch. The parser takes in a string corresponding to the expression assembled by the user. The parser recursively deconstructs the string in the order of operations, also taking in an evaluation value for x and for y, so that the code may return a number that corresponds to the value f(x,y) for the given x and y.

Derivative Deagle: The object was modeled in Unity from scratch. It has the functionality to select a point on the surface using a technique called ray casting, where a laser is emitted from the object and a coordinate triple is returned corresponding to the first point on the surface that the laser intersects. The user can also use the joystick to select the orientation of the curve, which mathematically corresponds to the direction of the directional derivative they would like to investigate.

Surface Generator: The surface is interpolated in Unity from a mesh of points in 3-space, the coordinates of which are calculated by sampling the plane on a grid and evaluating the z values by plugging the x and y values into the parser.

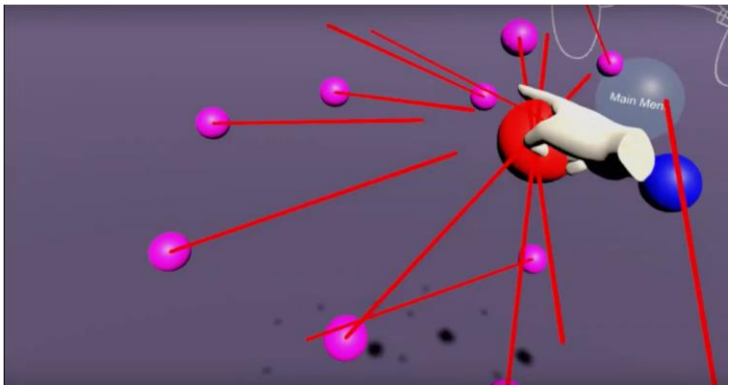
Deriver: The deriver, like the parser, takes in a string describing a function, and then recursively breaks it down and derives it part by part using the chain rule, product rule, sum rule, and other identities. It too must take in an x and y value, and then will return a numerical approximation to the derivative of the given function string evaluated at those x and y points. The numerical approximation is taken to several decimal points using the limit definition of the derivative, providing more than enough precision to make the model seamless.

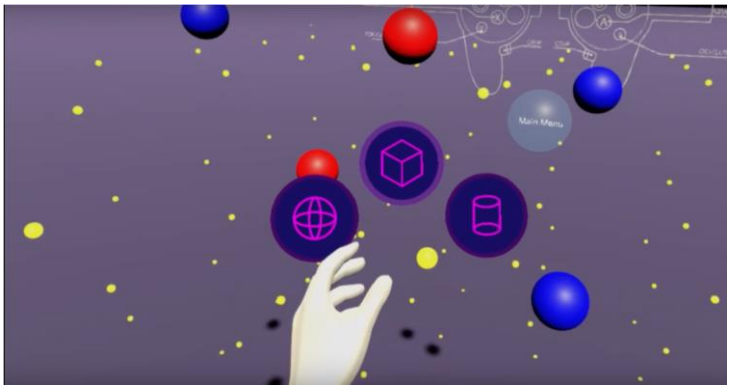
Geometry: We used cylindrical coordinates and some geometry to calculate exactly the equation of the curve the user would like to draw in terms of the variables the user controls, and in a format that Unity can use to render the curve.

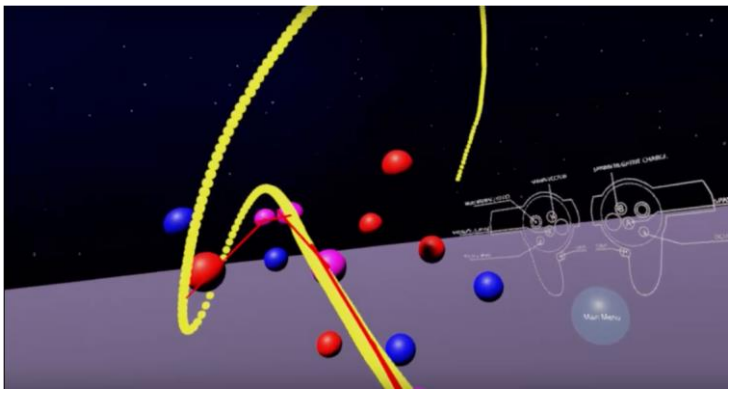
Curve Rendering: The final step is to draw the curve on the surface using a series of small balls to create the image of a dotted curve, which can be thought of as a cross section perpendicular to the xy plane, along which the directional derivative of choice can be thought of as a 1-dimensional derivative.

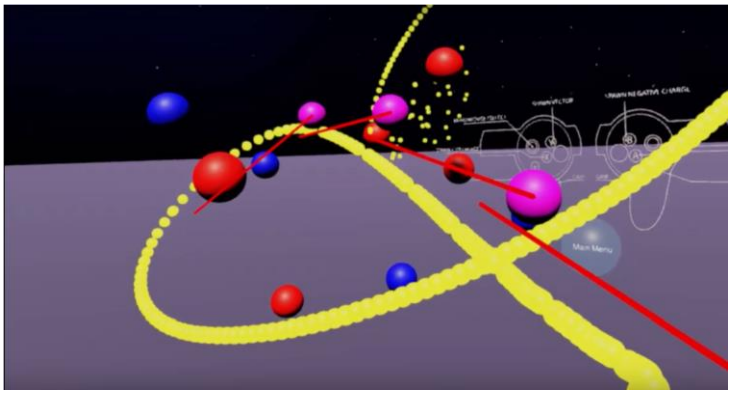
Electromagnetism Simulation

In the electromagnetism simulation the user is given the ability to place arbitrarily many positive and negative charges floating in 3-space. They can place test vectors that point in the direction of the vector field generated by those charges. The user is also able to emit a stream of electrons that will follow the vector field generated by the charges, that is, they will attract to the positive charges and repel from the negative charges according to the inverse square law, with the vacuum permittivity constant tuned to make the scale tractable and interactive. Finally, the user may generate a field of test charges in either rectangular, spherical, or cylindrical coordinates that allow the user to see how the entire space reacts to the charges. This also models any noncompressible flow with the positive and negative charges being sinks and sources, respectively.









Vector Field Simulator

The vector field simulator allows the user to select from one of 13 preset vector field options, and then walk around in a 3-D rendering of that vector field. The user will see many vectors floating around them pointing in the appropriate direction, and with a color to indicate the magnitude. A feature that we were working on implementing towards the end of the time limit was to allow the user to hold up a test point that emits a

vector in the correct direction, and to emit a stream of particles, much like in the second part, that will follow the path of the vector field. We have since polished these pieces of the project.

Mathematica and Unity Update

Three days after our project, Mathematics released Version 12, and Unity released Mathematica integration. With the two pieces of software being able to communicate, much of what we implemented in the project would become much simpler to implement by using Mathematica to solve and derive expressions, render surfaces, amongst other aspects. Our project was in effect a prototype for this integration, which coincidentally came out directly after it would have been useful to us. We mention the timing of the release of these updates as someone who is familiar with these pieces of software may notice that there are, now, easier ways to implement parts of our project.