Freeform 3D sculpting using signed distance functions

James Karlsson (13203260)

Abstract

This project explores the feasibility of using non-polygonal geometry as a basis for a freeform 3D sculpting application under a paradigm of creating and carving mass at arbitrary points in space. A prototype 3D sculpting application was developed to assess this approach in terms of performance and user experience. The main divergence of this prototype from conventional 3D sculpting tools is that it allows for freeform painting of geometry. This is enabled by using a stack of signed distance functions (SDF) as a source of truth for the sculpted shape, with constructive solid geometry (CSG) boolean functions used to combine SDFs in union and subtraction operations. In practice, this should allow for efficient sculpting of 3D shapes without users being bound by the constraints of mesh topology, whilst also requiring no prior knowledge of 3D coordinates systems or computer graphics to use.

This new approach came with unique user experience and performance challenges that the prototype attempts to address. The primary issues of this approach were poor depth perception leading to imprecision when creating mass in an empty area of space, and low application framerates when rasterising the signed distance field to a polygonal mesh for previewing purposes. The issue of depth perception is mitigated through several measures, most notably by adding support for Virtual Reality (VR) headsets and motion controllers, which also had the benefit of allowing for a more precise control scheme. Performance for editing relatively complex sculpts was made satisfactory on modern desktop computer hardware through the use of caching and performant data structures. With the prototype demonstrating this approach's viability, further research into improving the performance and user experience of non-polygonal sculpting applications is encouraged.

Contents

Introduction

When an artist is tasked with creating a 3D asset from scratch, they will generally begin with some basic shape (referred to as a "primitive"), such as a sphere. This shape is traditionally represented through an approximated "mesh" of polygons and vertices. The vertices in this mesh are then warped to fit the artist's intended shape, either vertex-by-vertex or through proportional editing.

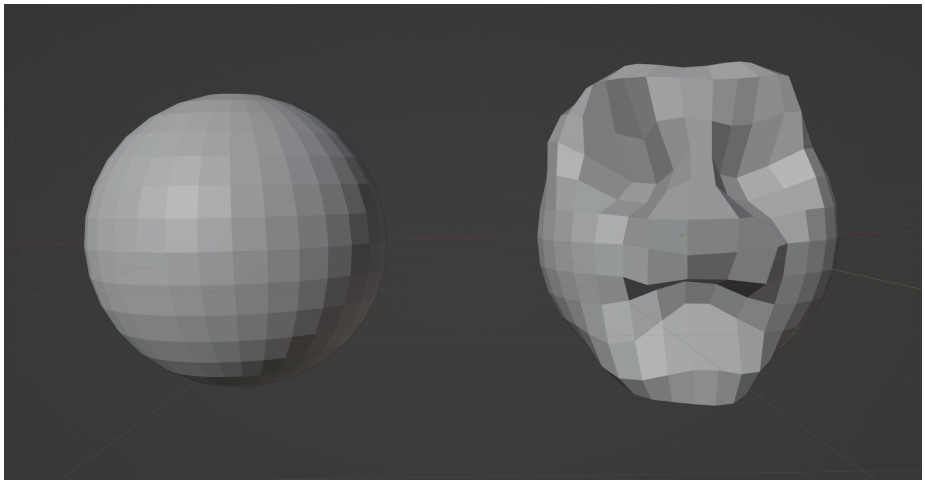

Figure 1: A sphere mesh in the modelling program "Blender" that has had its vertices sculpted to form a different shape. Each visible "square" (quad) represents four connected vertices.

Only a finite number of vertices can be used to approximate a given object as a mesh. Similarly, the way these vertices are connected - the mesh's "topology" - cannot change by simply shifting the positions of each vertex alone. Without tools such as "dynamic topology", level of detail at a given area decreases depending on how far apart vertices are from each other.

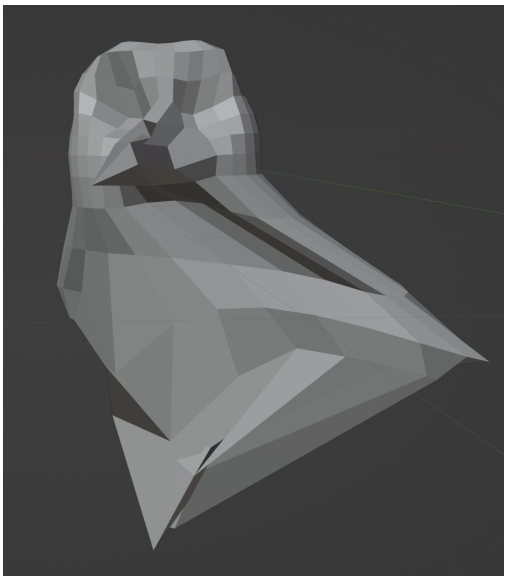

Figure 2: An attempt to pull the bottom of the shape downward. Note the decreased level of detail, to the point where adding further detail to that part of the mesh is impossible by moving vertices alone.

Dramatic changes to the shape of the mesh thus require manually creating new primitives at some defined point in space, and stitching them together, either through boolean operations (i.e. merging or subtracting two shapes) or by manually connecting the vertices between. This is a relatively complex process that requires prior knowledge of computer graphics concepts.

It is also possible for computer software to represent 3D shapes "implicitly", i.e. through a series of mathematical shape equations instead of through polygons and vertices. One such approach is through "signed distance functions" (SDFs). Signed distance functions represent a 3D shape by taking in a point in space (among other possible parameters) and returning whether that point was inside the shape (represented as a negative number), directly on the shape (zero), or outside of the shape (a positive number). For example, the signed distance function of a sphere would be $|P| - r$, where $|P|$ is the magnitude of the sphere's origin vector, and r is the radius of the sphere. A series of multiple signed distance functions can be combined to form a single composite signed distance function using constructive solid geometry (CSG) functions, allowing shapes to be added to and subtracted from

each other.

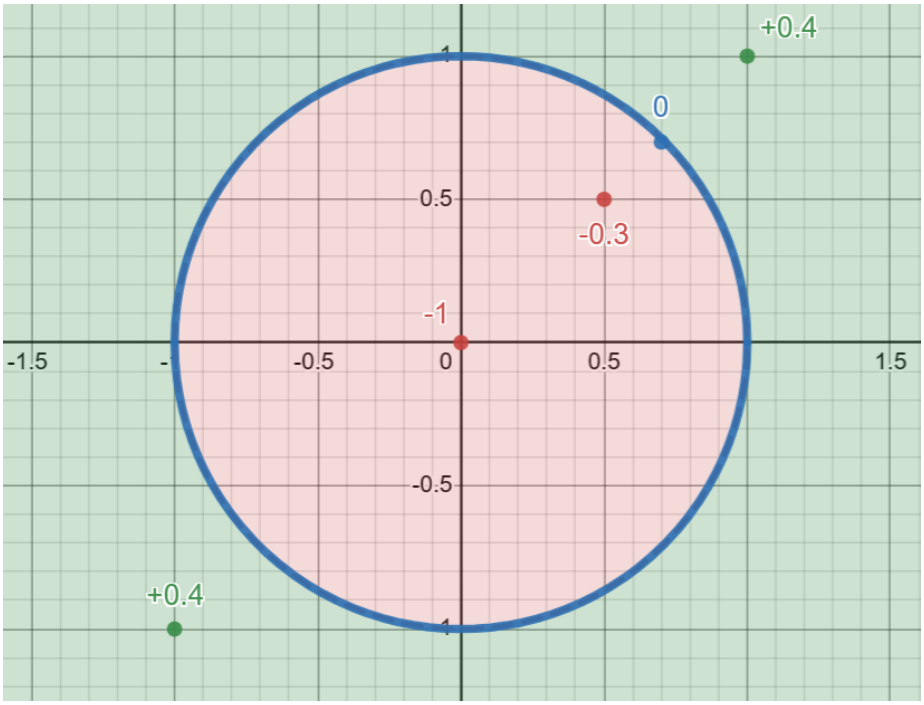

Figure 3: Plot of values for the signed distance function of a 2D circle, which behaves identically to values for a 3D sphere. Note that values outside the circle are positive, and values inside the circle are negative.

A strength of using implicit representations is therefore the ability to add or remove mass of arbitrary primitives at will. For example, cutting a hole through a sphere is simple to represent with signed distance functions, with a function like $max(-a, b)$, where a is the value of the signed distance function for the original sphere, and b is the value of the signed distance function for the object being cut into it. The same operation on a sphere represented as a polygonal mesh however would require dramatic changes to its topology. This complexity would generally be passed onto the user of a sculpting application, as moving the positions of vertices alone cannot modify topology. In theory, another strength of implicit representations is the ability to preserve fine details at arbitrary resolutions; for polygonal sculpting, this is something that would require dynamic topology or multi-resolution mesh system to replicate.

This project explores the viability of basing a freeform 3D sculpting application on

the principles of creating and carving mass at arbitrary points in space, as opposed to moving vertices on a polygonal mesh. This is enabled by using a stack of signed distance functions as the internal representation of the shape being sculpted.

The implementation of this sculpting application thus diverges strongly from most 3D sculpting tools, which solely rely on a polygonal representation for a mesh, as well as most existing CSG-based modelling tools, which require manual placement of primitives and setting of parameters. In this context, the term "freeform" was used to emphasize the absence of rigid constraints, enabling users to shape objects in a fluid manner by effectively drawing in 3D space, rather than being bound by traditional technical intricacies of 3D modeling.

In addition to being controllable with a keyboard and mouse and on a traditional computer monitor, support was also added for VR (Virtual Reality) headsets such as the Meta Quest and their motion controllers. This was to explore how factors such as stereoscopic rendering and the additional set of natural analog inputs could be used to more intuitively and accurately manipulate the model, as well as give the model a stronger sense of physicality. Models could also be exported to a polygonal mesh for use in programs that rely on that representation, such as 3D printing, rendering engines, and even traditional modelling software for further adjustments.

Research was conducted through a combination of theoretical and applied research, under the paradigm of design science. Design science focuses on the development and evaluation of designed artifacts, and is a well-established paradigm for research in engineering and computer science (Vaishnavi & Kuechler, 2015).

Source code, compilation instructions, videos of the application in use, and precompiled Windows builds of the application have been made available at https://github.com/jaaamesey/SUSMA.

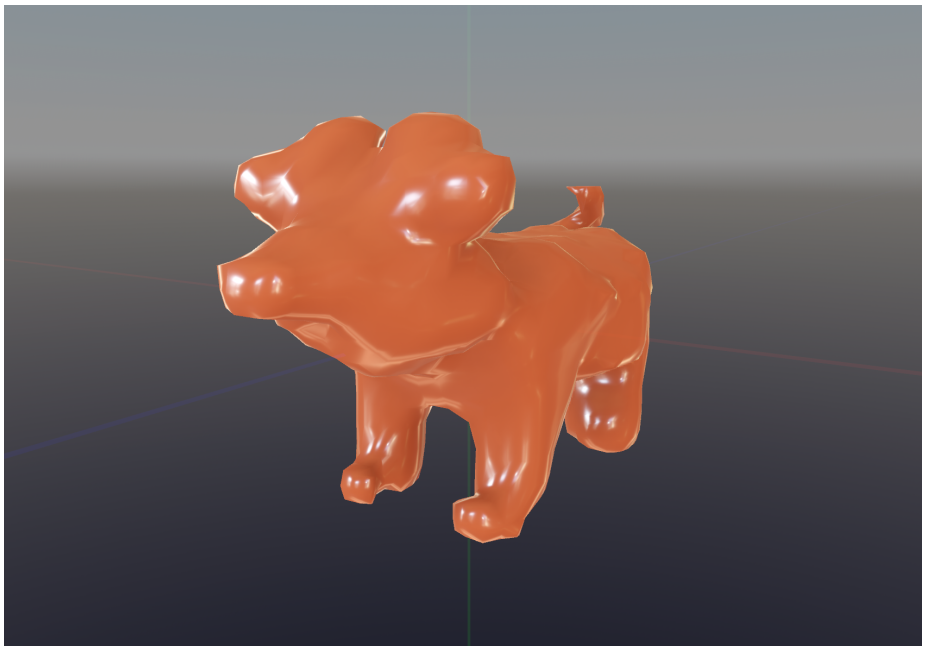

Figure 4: A small sculpt created in roughly five minutes using the final version of the application.

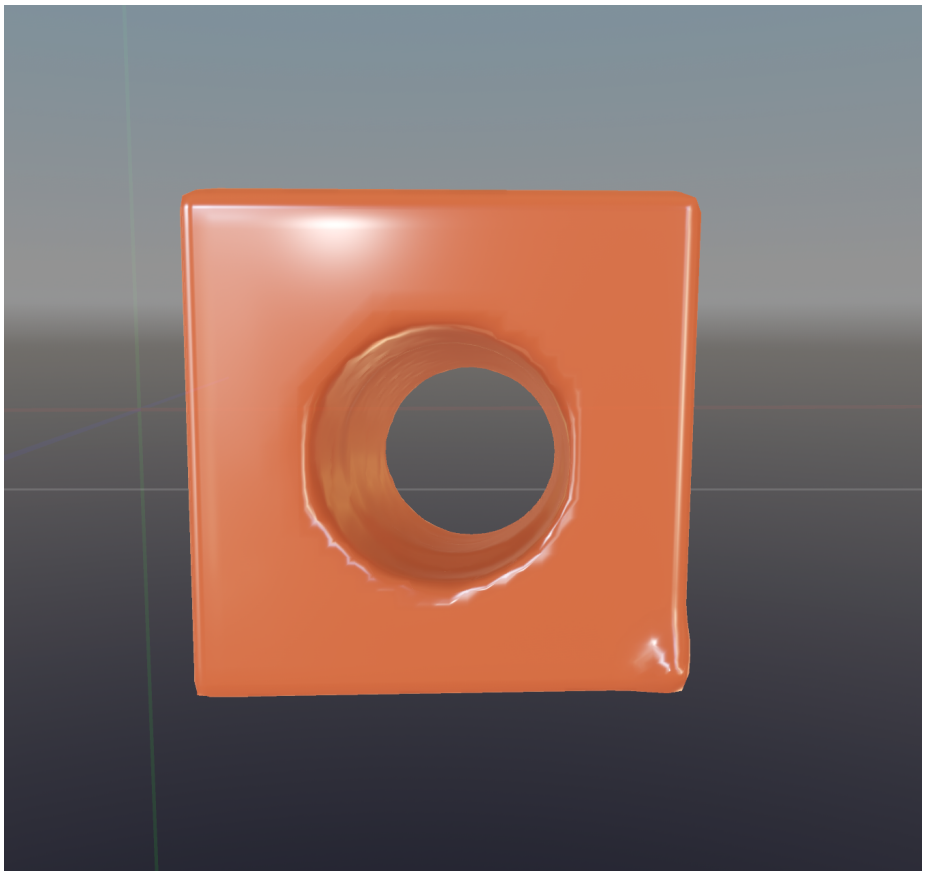

Figure 5: A cube that has had a hole cut into it, demonstrating that sculpts created in this application are not bound by fixed topology.

Literature Review

The modelling programs “ZBrush” and “Blender” can make use of dynamic topology systems to add mass when the current polygon count in an area is deemed insufficient. However, as mentioned, making dramatic changes to a polygonal mesh’s topology, such as cutting a hole into part of it, is not a spontaneous process.

With regard to using implicit surfaces for sculpting, one challenge is how to render and display such a surface efficiently and accurately. A common approach is to use polygonal meshes to render an approximation of an implicit surface, leveraging more traditional rendering techniques (Sigg, 2006). However, to achieve the level of responsiveness expected by a commercial sculpting application, this would inherently require a meshing system that is performant enough to run in real time, upon each operation applied to the underlying model.

An alternative approach is to use ray tracing or ray casting techniques to directly render the implicit surface without using any intermediate representation. For example, methods currently exist to directly render non-polygonal objects on graphics hardware using vertex and fragment programs (Ii, 2004). There are several examples available on platforms such as Shadertoy that demonstrate these techniques running in real time on contemporary graphics hardware (YVT, 2016) (FAD, 2022), as well as the work of Söderlund (2021). The video game “Dreams” also renders shapes as signed distance fields with no intermediary polygonal representation, instead rendering out to point clouds (Evans, 2015).

Another method to represent shapes implicitly is through neural implicit surfaces, which involves encoding a surface as a level set of a neural network applied to spatial coordinates. Techniques have recently been introduced to substantially increase the performance of neural signed distance functions, making them more practical for real-time graphics applications (Takikawa et al., 2021). Sharp & Jacobson (2022) also introduced an approach to perform geometric queries such as ray casting on neural implicit functions via range analysis with concrete accuracy guarantees.

Point clouds are also a widely used non-polygonal representation of 3D shapes, with techniques available to at least export them to a clean polygonal mesh through surface reconstruction (Berger et al., 2016). Pauly (2003) demonstrates a way to perform common sculpting operations on point clouds through an application called Pointshop3D. For example, "embossing" and "engraving" operations can be performed by applying normal displacements on the sample positions, and a "carving" operation can be achieved through boolean intersection. Despite its apparent age and complexity in terms of user experience, this application demonstrates that at a technical level it is possible to sculpt point clouds in a very similar way to a traditional polygonal mesh.

With regard to how implicit surfaces can be manipulated by a user in a way that mimics commercial sculpting applications, existing research in this area is more limited. A method for deriving operators from user sketches for blending multiple implicit surfaces together has been demonstrated (Angles et al., 2017). Additionally, Du & Qin (2000) demonstrated interactive sculpting of PDE (Partial Differential Equation) surfaces, though this appears to some extent to still depend on a fixed grid resolution. Reiner et al. (2011) described a modelling approach using signed distance functions, capable of being rendered directly in real time through a generated fragment shader and sphere tracing. Shapes were able to be manipulated in complex ways to form objects in a relatively intuitive way, requiring no understanding of the underlying mathematics.

A more recent example of a 3D modelling program making use of implicit surfaces is MagicaCSG (ephtracy, 2021). The program allows for 3D models to be created through the use of constructive solid geometry. Models are built by explicitly defining the primitive shapes that make up an object and defining how they should be blended together. Another tool by the name of "QuickCSGModeling" (Chen et al., 2023) follows a similar paradigm, but makes use of a VR headset for more intuitive navigation around the model and more intuitive and granular cursor movement.

Internally, both applications make use of signed distance functions to represent shapes and how they are blended together. Both applications also handle previewing the

model by rasterizing the signed distance field to a mesh at a fixed voxel size.

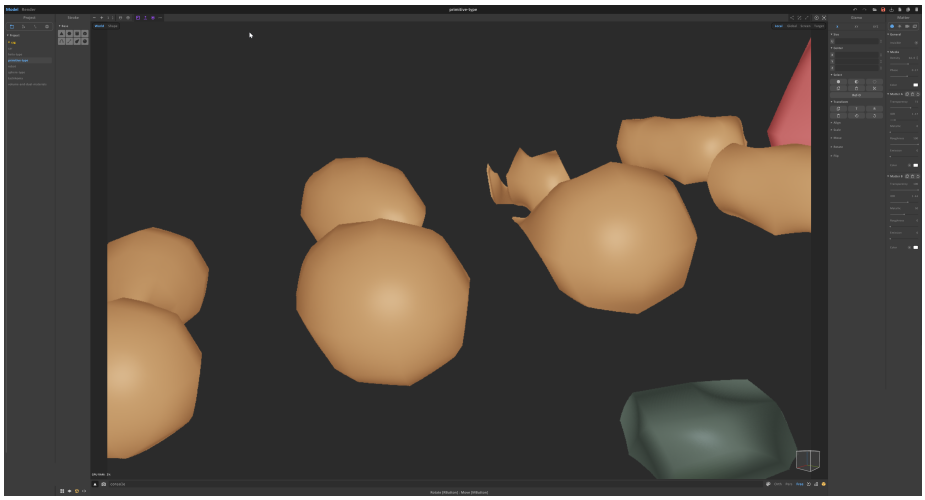

Figure 6: MagicaCSG rasterizing SDF shapes to a low voxel resolution.

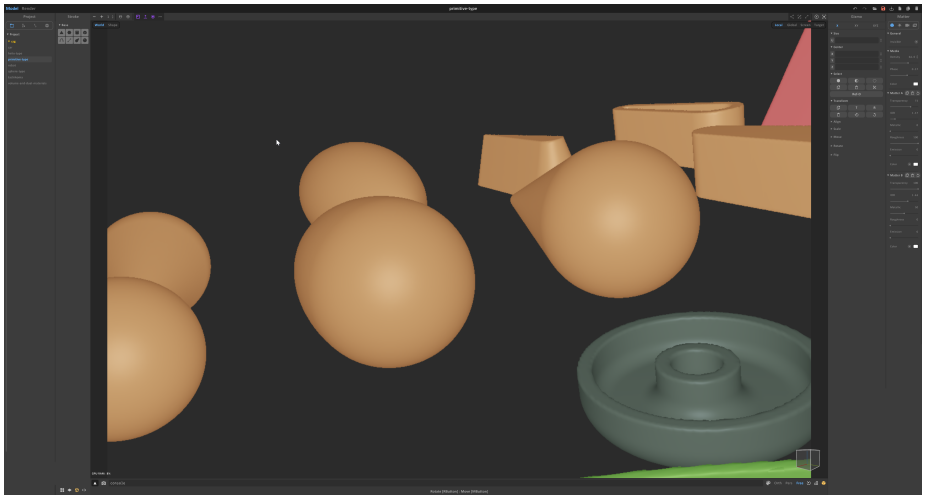

Figure 7: MagicaCSG rasterizing SDF shapes to a high voxel resolution.

The approach used by MagicaCSG and QuickCSGModeling lends itself exceptionally well to hard surface modelling. However, more organic objects such as human faces are more difficult to represent in this manner, as there is no equivalent to the more freeform sculpting exhibited by programs such as ZBrush.

Current gaps in research

Not much recent research into using implicit surface representations for organic 3D sculpting could be found, despite an abundance of such techniques being used for hard surface modelling.

The main challenge of using implicit surfaces specifically for organic 3D sculpting applications appears to be how to create and modify the underlying scalar field that defines the shape in response to natural user input. This also appears to be the primary gap in research in this area. A common approach is to use a set of control primitives, such as points, lines, or planes, that generate local field functions and combine them with some blending function. However, these methods have some limitations, such as requiring manual placement and adjustment of the control primitives, and having difficulty in creating sharp features or fine details. This has been explored in research to some degree, with findings that a classical implicit representation for sculpting would lead to a "complexity explosion" due to the increasing number of primitives, with iterative shape refinement being a key aspect of sculpting (Ferley et al., 2000).

Much of the research in this area is quite old (decades old in some cases), with the landscape around graphics and computer hardware changing substantially since; an example with this is with the rising practicality of real time ray tracing (Deng et al., 2018), which might mitigate concerns regarding rendering performance when a sculpted object needs to be visualized at a high quality. Additionally, despite the work of Pauly (2003) demonstrating real-time sculpting of point clouds, there is room to target a user experience more in line with modern sculpting applications. Thus, there would likely be some benefit to revisiting prior literature from the perspective of this new landscape, taking into account changing performance characteristics, user experience

expectations, and any novel techniques introduced since.

Method

Each modification to the model was represented through an "operation" object that was simply appended to a single stack, which acted as the single source of truth for the final shape of the model. Key attributes of these operations included a type (e.g. whether it was an "ADD" or "SUBTRACT" operation), a primitive shape (e.g. a sphere or cube), and the 3D position in space where the operation occurred at. These operations would then effectively be converted to a signed distance function, which would be combined with prior operations through a blending function (either a smooth CSG union or difference function). These operations would be continuously added on top of each other as the user moved the cursor, which allowed for freeform brush strokes. For performance reasons, primitives were not simply added to the stack each frame a mouse button or trigger were held down: the cursor had to be moved by a small amount before a new primitive could be placed. This amount was tuned in order to balance performance and the perceived smoothness of the brush stroke, and scaled proportionally with how zoomed out the viewport camera was from the origin.

```
### GDScript code for determining whether to place a new primitive
if last_held_brush_pos == null \
    or brush_pos.distance_squared_to(last_held_brush_pos) \
        > (0.0001 * camera.position.z):
    # Place new primitive
```

Shape and blending functions were largely derived from an article by Quilez (n.d.) that contained reference implementations of signed distance functions in GLSL. The use of factor k in blending functions allowed for control over how sharply an operation would blend with the rest of the shape. Whilst controllable by the user, care

was taken to ensure a sensible default value, which gave the appearance of brush segments transitioning into each other naturally without unacceptably warping nearby geometry. After some iteration, this value ended up being 0.5 multiplied by the radius of the applied brush in world units.

```
/// C++ implementations of smooth union and subtraction functions
double opSmoothUnionSDF(double a, double b, double k)
{
    double h = clamp(0.5 + 0.5 * (a - b) / k, 0.0, 1.0);
    return lerp(a, b, h) - k * h * (1.0 - h);
}

double opSmoothSubtractionSDF(double a, double b, double k)
{
    double h = clamp(0.5 - 0.5 * (a + b) / k, 0.0, 1.0);
    return lerp(a, -b, h) + k * h * (1.0 - h);
}
```

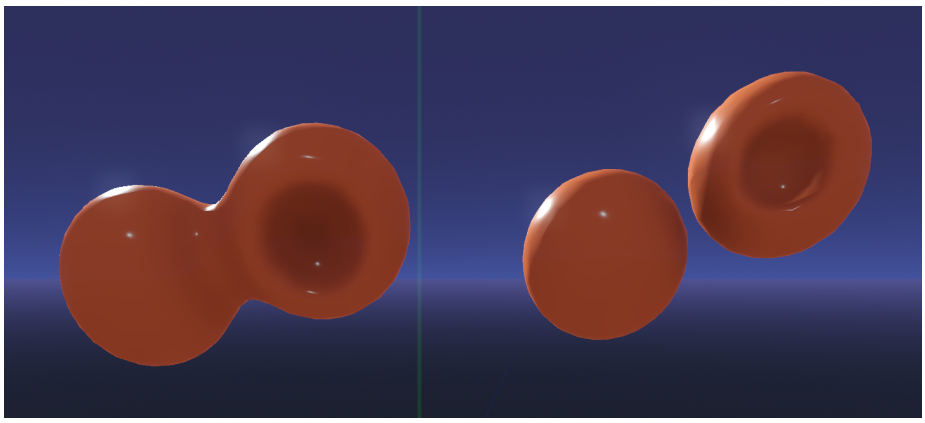

Figure 8: High (left) vs low (right) blending factor.

Two approaches were examined more thoroughly for rendering the realtime preview of the sculpted shape: rendering the shape directly through ray tracing, versus converting it to a mesh, with the latter proving more promising at the time. It was hy-

pothesised that running increasingly complex signed distance functions on the GPU would not scale as efficiently as rendering an equivalent rasterised polygonal representation. Rendering a polygonal representation was also assumed to be less time-consuming due to OpenVDB having built-in functions for such a task, with it likely being expected that any sculpting program should always be able to output to a standard mesh format anyway.

OpenVDB was used to convert the signed distance field to a fixed resolution level set grid, which was then converted to a polygonal mesh; this mirrors the approach used by MagicaCSG and QuickCSGModelling. Godot Engine 4.1 was used to render a real time preview of this mesh, and was also used for rendering a basic user interface and handling user input. The Godot Engine was chosen over engines such as Unity and Unreal Engine as it was relatively lightweight and allowed for integration with C++ libraries (required for interfacing with OpenVDB), whilst still offering a suite of tools that lended themselves well to rapid prototyping.

Code that interfaced directly with OpenVDB or was otherwise performance-critical (e.g. computing vertex normals) was written in C++ using the GDExtension API, with the remainder of the code being written in Godot's bespoke scripting language, GDScript. This was to achieve a solid balance between performance and prototype iteration time.

The GLTF format was used when exporting the rendered mesh. Existing polygonal meshes could also be imported using the same GLTF format, with vertices of the input mesh crudely converted to spheres that were blended together.

Control scheme design

The control scheme for both the traditional mouse and keyboard version of the application and its VR version were iterated upon heavily throughout the course of the project's development. Operations expected to be performed commonly were placed close together in easily reachable places: this included creating and carving mass,

Figure 9: Blender “Suzanne” model imported and converted to a signed distance field.

adjusting brush size, and moving and rotating the brush cursor. Less frequent actions such as toggling symmetry could thus be moved to more obscure inputs, such as reaching for the TAB key or clicking in the right stick. The use of the W and S keys for moving the brush forwards and backwards mimics that of movement in contemporary 3D applications, whilst also making sense ergonomically; the mouse is still usable in the right hand, with the position of the left hand still being close enough to other common inputs.

Modern VR controllers differ from keyboard and mouse setups in that despite having a much larger number of analog inputs (such as analog sticks and positional/rotational tracking), they only have a small pool of buttons to assign actions to. This required a higher reliance on chorded inputs, such as holding the left controller's X button and tilting the right controller's stick to adjust brush size. With the brush size controls serving as an example again, chorded inputs involving face buttons would always make use of the analog stick of the *opposite* controller for ergonomic reasons, as typically only one finger would be comfortably available in the area those inputs sit. This was not an issue for triggers at the back as a dedicated finger would always be resting there.

With the mouse and keyboard control scheme not having an equivalent to rotating the right VR controller to affect the rotation of the brush (applicable to the cube and cone brushes specifically), this action was instead replicated by holding the "R" key and using the mouse's X and Y velocity to adjust the Y and X Euler rotations of the brush respectively. A similar setup was later ported to the VR control scheme, allowing for larger rotations of the brush to be achieved without making potentially uncomfortable hand movements.

Management of depth perception

Representing shapes implicitly allows for mass to be easily created and destroyed at arbitrary positions in space; those using the application no longer have to consider the topology of new meshes being added to a scene, they only have to consider the mesh's

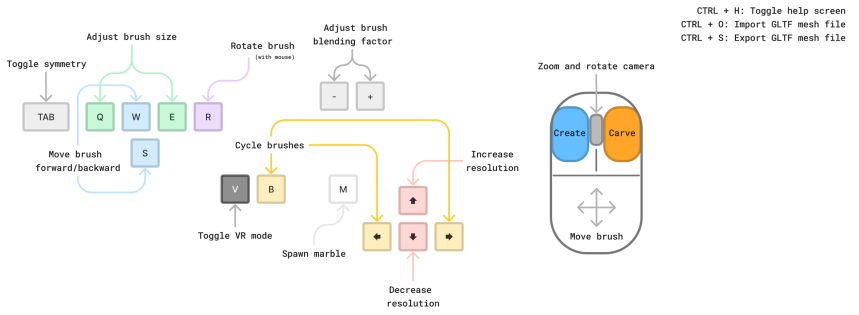

Figure 10: Mouse & Keyboard Control Scheme for the application.

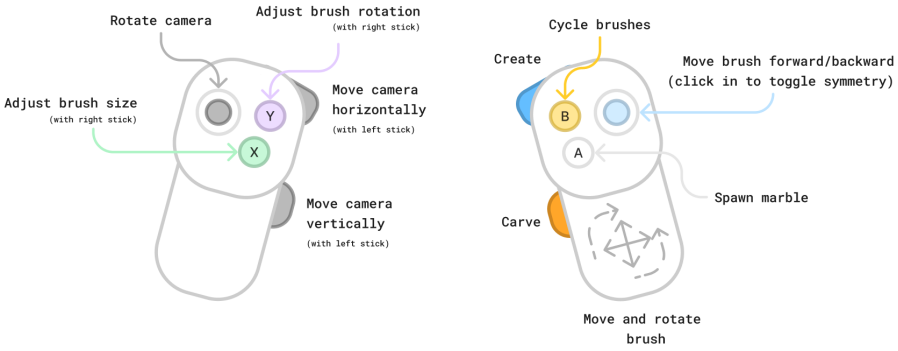

Figure 11: VR Control Scheme for the application.

location and shape. If we want to consider adding and removing mass as a natural, intuitive part of the sculpting process, the user must be able to easily determine where their cursor is sitting in 3D space.

ZBrush handles this by drawing a circle facing the normal of the vertex the cursor is currently sitting on. However, the cursor's distance from the camera is always locked to whichever vertex is closest on that axis - there is no way to move the cursor forwards and backwards, only horizontally and vertically. The limitation of the cursor always following the existing mesh thus makes it much simpler for a user to determine its position.

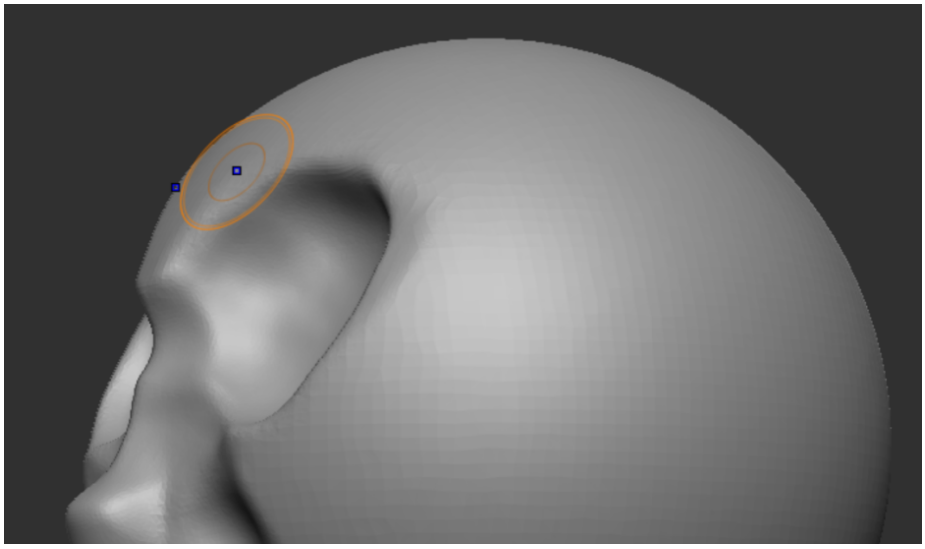

Figure 12: ZBrush's cursor when hovered over part of the mesh.

Given the inherent visual and conceptual complexity of having a cursor that can move forwards and backwards relative to the viewport, the following measures were put in place to aid depth perception, listed in order of their implementation:

1. Adding a crosshair and fixed size dot to the cursor preview, to make it easier for the eye to distinguish between its size and distance to the viewport camera.
2. Adding a horizon to the background of the application, making it easier to determine how the camera is oriented in terms of pitch.

3. Adding support for VR headsets, with distance to the viewport camera communicated through stereoscopic rendering and parallax upon natural head motion.
4. Including the cursor's depth from the camera (or right controller in VR) in the top left info panel.
5. Applying haptic feedback to the right VR controller when the cursor collided with solid parts of the signed distance field. This also gave an additional sense of physicality to the sculpt.
6. Adding a small point light to the cursor, making it easier to gauge how close the cursor is to existing parts of the shape.
7. Adopting a more natural lighting setup (including the use of an environment map), making features along the surface of the model more distinguishable.

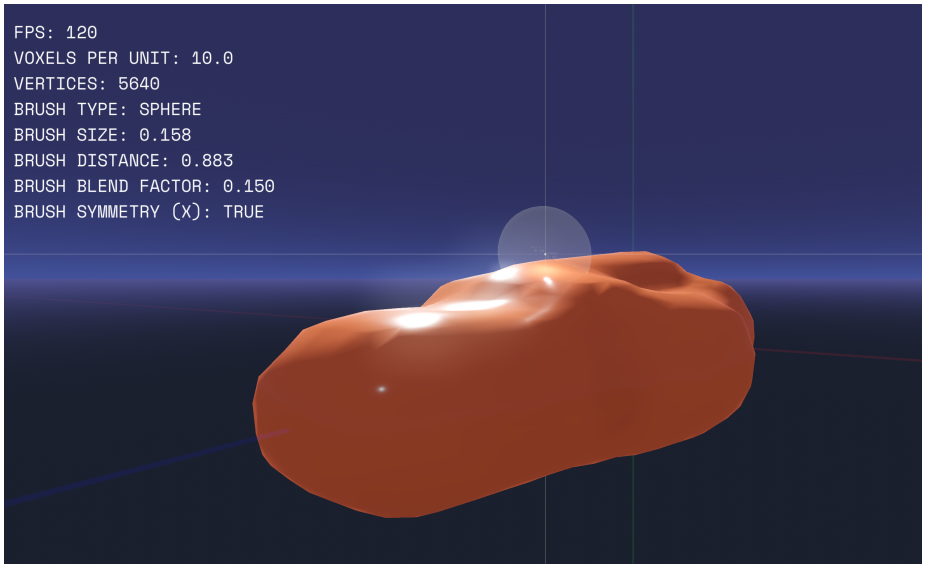

Figure 13: This application's cursor.

Ethical implications

Overall, the ethical implications of this research were relatively minimal - no test subjects were involved, and no personally identifiable information was collected. One potential area of concern was accessibility - care was taken to ensure that sculpting operations on a non-polygonal representation of an object can be performed to a

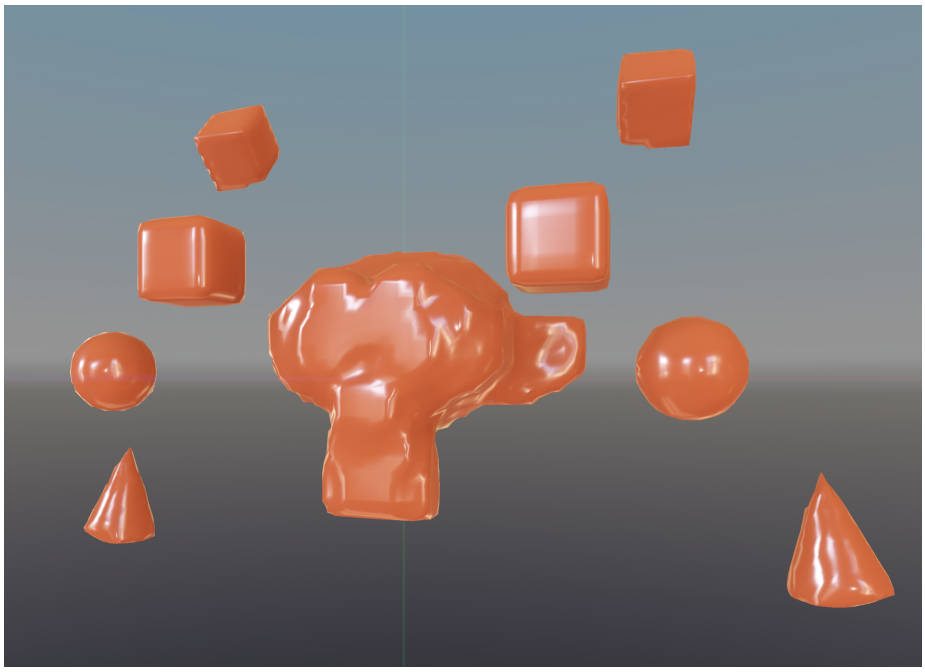

Figure 14: Final lighting model.

comparable level of ease to commercial sculpting applications, with the impact of this mitigated by making it clear that the purpose of any produced artifacts would only be to demonstrate the viability of basing a sculpting program on non-polygonal shapes, with the artifacts themselves not being intended for widespread use.

Regardless, any software created would have been expected to be as safe, stable, and energy efficient as practical, as distributing software in any way would come with the expectation that any device may run it. Existing code and test models used also received proper attribution, with it being crucial to properly attribute and acknowledge any assets, code snippets, or resources used from external sources. Compliance with licensing agreements and copyright laws ensures that creators and rights holders are duly recognized for their contributions and that the project adheres to legal and ethical standards. Some level of continuous reflection and evaluation was undertaken to identify and address any unanticipated consequences or ethical dilemmas that might emerge; however, as mentioned, the ethical impact of this project was identified as fairly low risk.

Findings

Three hardware configurations were tested to gauge the program's performance on contemporary desktop computer hardware:

1. High-end desktop: AMD Ryzen 7 7800X3D, NVIDIA RTX 4090
2. Mid-range desktop: AMD Ryzen 5 2600, NVIDIA RTX 2060
3. High-end notebook laptop: AMD Ryzen 7 PRO 7840U, AMD Radeon 780M (integrated GPU, tested on battery with power saving mode)

Vertex count served as a surprisingly reliable heuristic of a model's complexity at a given voxel size. In practice, it had the most reliable correlation with frame rate during manipulation of the model when compared with metrics such as number of operations and the overall bounding box of the signed distance field.

It was found that on all three configurations, relatively complex models in the hundreds of thousands of vertices could be rendered without issue, at well above 60 frames per second. When re-meshification was required however, i.e. when the model was modified, mild stuttering would occur. The intensity of this stuttering scaled linearly with the vertex count of the rendered mesh. Temporary drops to 15 frames per second would occur at approximately 180,000 vertices on the high-end notebook laptop configuration, and 400,000 on the high-end desktop configuration; drops to 30 frames per second would occur at only 70,000 vertices and 200,000 vertices respectively.

This stuttering was made less perceptible in VR through asynchronous reprojection (Developers, 2018). Similarly, outside of VR, the operating system's cursor would still render at the display's refresh rate, with camera movements unable to occur during manipulation of the model. Regardless, 15 frames per second was the approximate framerate where the fluidity of brush motions would start to be compromised. According to Nielsen (2019), 0.1 seconds is the point at which a user no longer feels a system is reacting instantaneously; in this context, a framerate below 10 frames per second would thus be considered unacceptable.

With shapes being stored losslessly as signed distance functions, in the event of meshification introducing unacceptable levels of stuttering, there was also the option of lowering the voxel size to more smoothly make changes to the model, and increasing it later without any loss of the original detail.

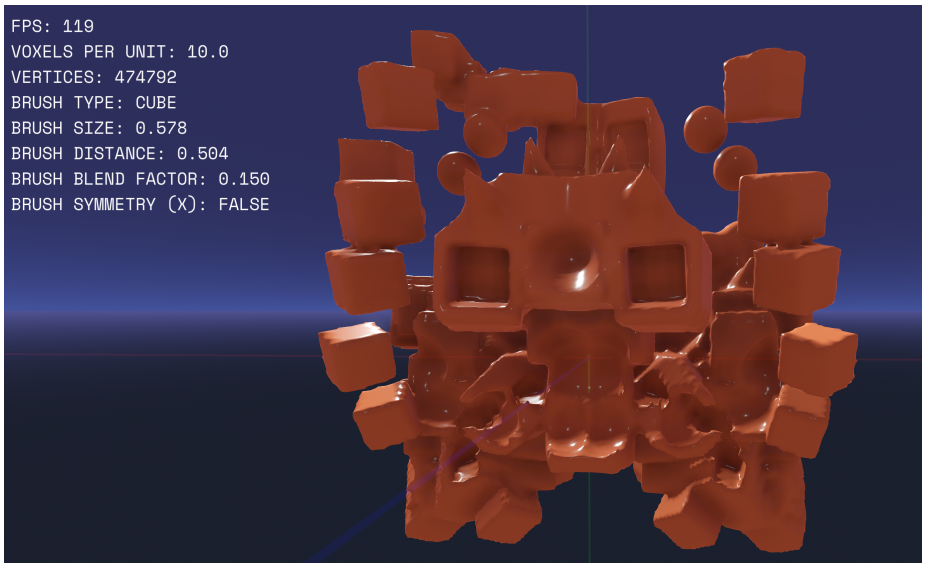

Figure 15: Example of a complex, half-million vertex model created in the application on the high-end desktop configuration. Significant modification beyond this point would require a decrease in voxel size for performance to be perceivably smooth.

It was found that explicitly setting the "background value" - the value used by OpenVDB for parts of the grid not yet specified - to 1.0, as opposed to the default value of 0.0, significantly improved the quality of the rendered mesh, reducing perceived aliasing. It's assumed that this is due to signed distance values of zero corresponding to a hard edge, which would have likely affected interpolation during meshification.

Performance was increased dramatically by reusing the previous state of the signed distance field grid between operations, instead of re-computing every previous operation on the grid each time a new operation was added to the stack. Complex sculpts would have otherwise been practically impossible to produce - framerates during manipulation of a 400,000 vertex model at 10 voxels per unit were measured at less than 1 frame per second as opposed to the aformentioned 15 on the high-end desktop configuration when this caching was disabled. However, this did not improve performance

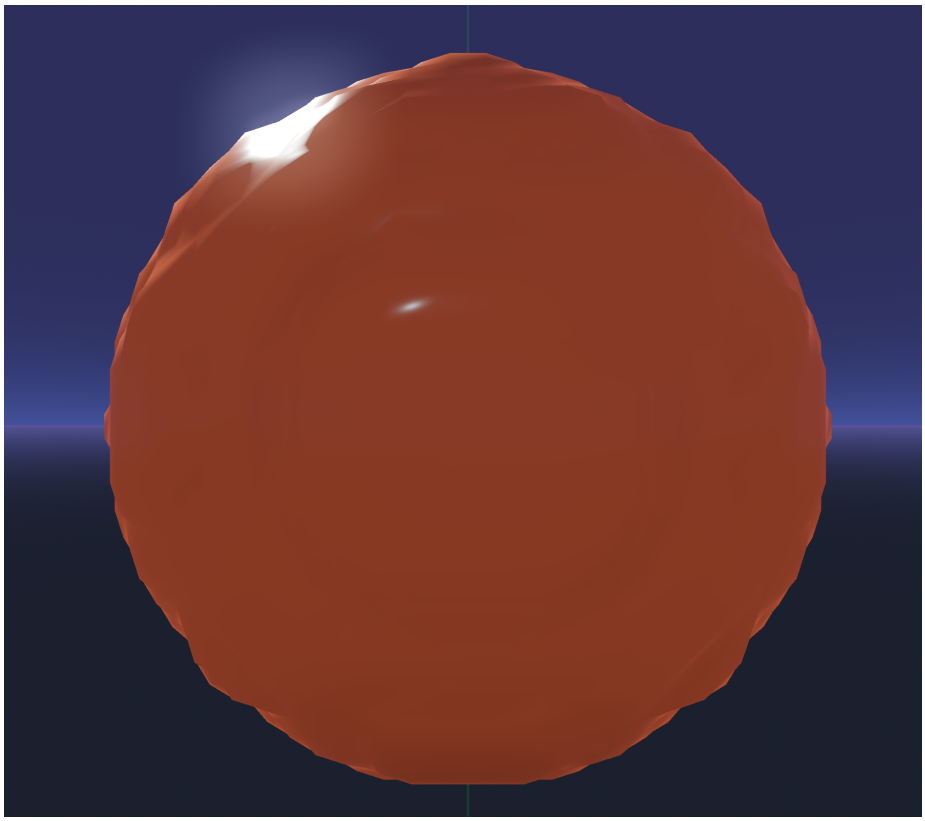

Figure 16: Background value of 0.0 applied to grid.

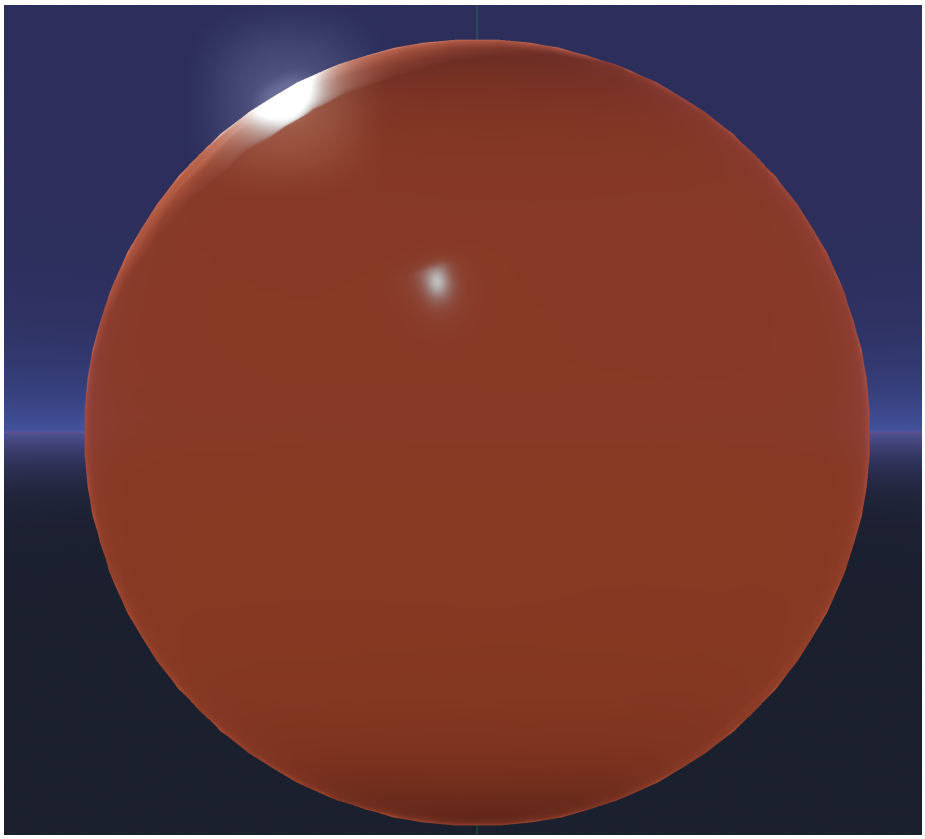

Figure 17: Background value of 1.0 applied to grid.

when adjusting voxel resolution or loading an initial mesh from a file, as these actions required recomputation of all prior operations.

Performance also increased substantially when assigning each operation its own bounding box on the grid to iterate through, instead of always recomputing the entire grid at some fixed size. As a side effect, this also allowed for the canvas to effectively become infinite, as it removed the need to set boundaries for the canvas in the first place.

With these performance optimisations in place, the final conversion of the OpenVDB grid to a mesh became the primary bottleneck, having a significant practical impact on user experience due to needing to occur for each frame the user was modifying the model. This scaled exponentially worse with increases in voxel resolution, and linearly worse with the number of defined values on the grid, but was not impacted much directly by number of operations, size of the grid, or even final polygon count.

However, despite the theoretical advantage of implicit surface representations being resolution agnostic, the reality of converting to a mesh resulted in finer details (generally those less than 0.05 units wide) not being visible in the preview without an increase in the number of voxels per unit that would cause severe stuttering as the complexity of the signed distance field grew. With no equivalent to ZBrush and Blender's dynamic topology systems, detail in the voxel grid and consequently the rendered mesh could not be localised to only specific regions of the model; in other words, the model was constrained to being rendered at a constant level of detail.

Discussion

The final prototype demonstrates the viability of building a 3D sculpting application that uses implicit surfaces as an internal representation of a sculpt, specifically with signed distance functions.

[1] This purely demonstrates the loss of detail at lower voxel resolutions, not a specific situation where a higher voxel size would massively degrade performance.

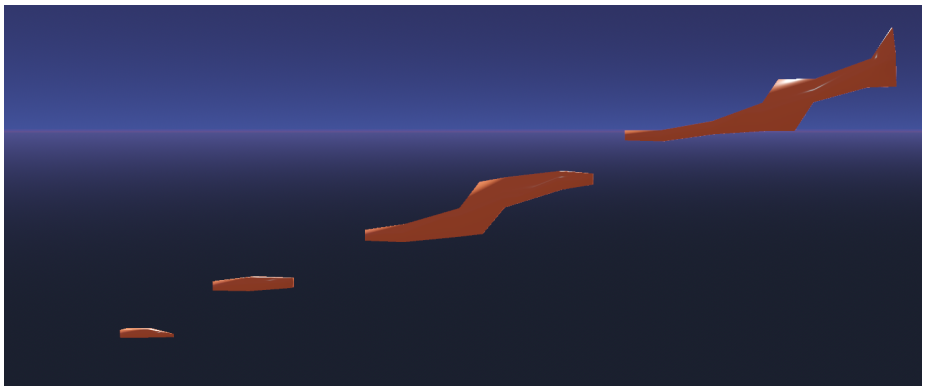

Figure 18: Brush stroke of size 0.043 at 10 voxels per unit[1]

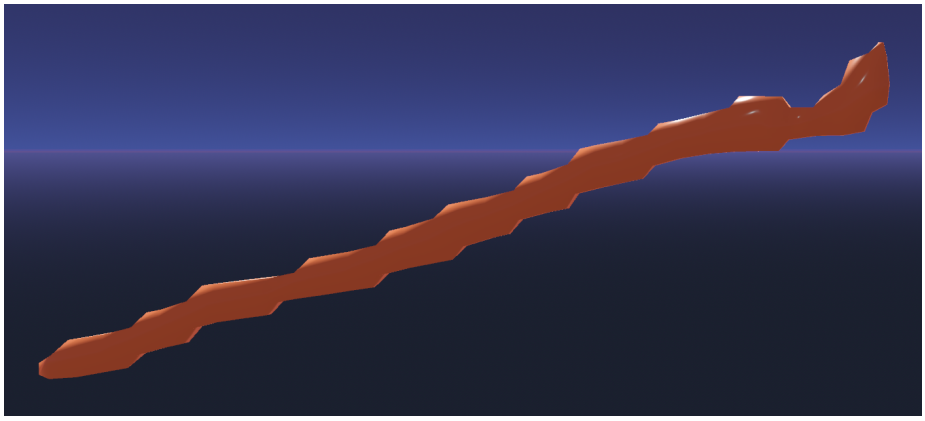

Figure 19: Brush stroke of size 0.043 at 20 voxels per unit.

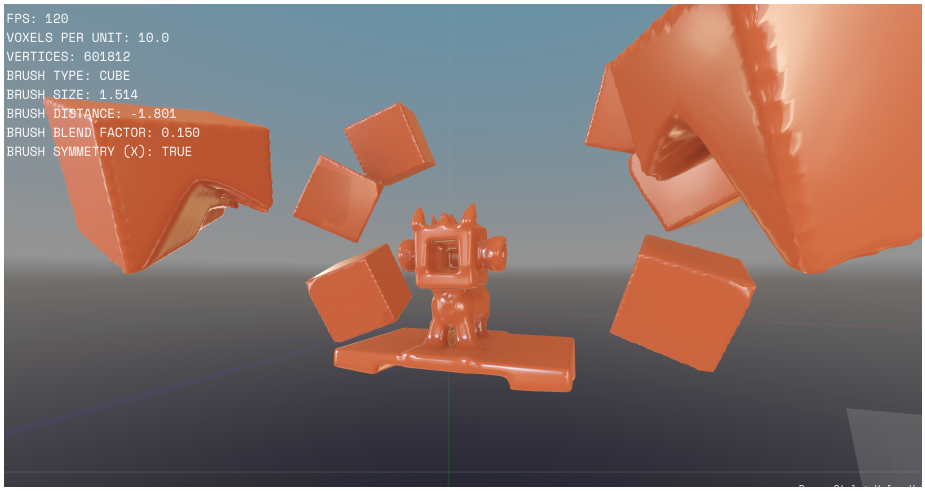

Figure 20: A moderately complex sculpt created in the application.

It was initially thought that the primary benefit of using implicit surfaces for sculpting would be the ability to have a lossless, resolution-agnostic representation of a 3D model. However, what ended up being of more interest was the speed at which shapes could be created and carved without being bound by topological constraints. It is assumed that those constraints are not much of an obstacle for professionally trained users of polygonal sculpting programs, but are instead constraints worth eliminating for users less familiar with 3D sculpting and computer graphics concepts in general. Additionally, the ability to export to a common polygonal mesh format enables a high degree of versatility to objects created in the application, allowing them to be 3D printed, shared online, put into a game engine, or even used as a base for a more detailed model in another modelling application.

Signed distance functions proved themselves to be an efficient medium for expressing shapes implicitly. The primary performance bottleneck was instead being the process of transforming the rasterised grid to a polygonal mesh, contributing to stuttering when more ambitious sculpts were modified. The ability to modify voxel resolution at will mitigates this issue somewhat, but ideally this would be something a user would not need to think about. Future sculpting applications making use of signed distance fields or other non-polygonal representations should thus consider performance improvements that substantially decrease re-meshification time.

Depth perception remains an important issue with a modelling paradigm based on creating and carving mass at arbitrary points in 3D space, particularly with regard to the former. Whilst significantly mitigated in VR where stereoscopy is available, the likelihood of users creating mass at undesired positions becomes a more significant issue when the application is viewed on a standard 2D monitor. Additionally, VR controllers were found to allow more precise and intuitive control than a mouse and keyboard due to the wider range of natural analog inputs. In addition to improvements that would directly aid depth perception, future applications relying on this paradigm should ideally have tools that can reduce the severity of this inherent imprecision, such as being able to easily shift the position of matter after it has been created.

Overall, whilst the final project allows for rapid freeform creation of organic 3D objects at a moderate level of detail, with acceptable performance on modern desktop hardware, there is definitely further room for further research and experimentation in making such an approach more viable, both in terms of performance and other usability factors.

Potential performance improvements

There are numerous performance improvements that could still be made to make a sculpting tool based on signed distance functions more practical for representing fine details and larger objects. These same improvements would also likely increase the viability of running such a tool on lower end hardware, and make the average user experience less impeded by stuttering.

Meshification in theory only needs to modify vertices at point of change. Despite modification to the voxel grid being roughly localised to the region of change, the entire mesh is still recomputed from scratch from the state of the entire voxel grid. This results in a large amount of redundant computation, and likely remains the most significant performance bottleneck.

Using some kind of multi-resolution grid for meshification, instead of using a uniform grid with a fixed voxel size, may also aid performance. In theory, large brush strokes could be stored at a lower voxel resolution than finer strokes. Finer details could also be represented using normal/bump maps, instead of requiring extraordinarily high polygon counts to be visible.

Rendering the model without the use of a mesh as an intermediary representation also warrants further investigation, given that meshification proved to be the most significant performance bottleneck during the development of this tool. This could be achieved through rasterising to point cloud particles instead, similar to the work of Evans (2015), or through real time ray tracing.

The internal representation of brush strokes could also be changed from a large series

of points to a representation such as Bezier curves, for less memory use and shorter meshification times from scratch, i.e. when adjusting voxel size or loading a file. This would more closely align with how brush strokes are stored for vector-based 2D drawing applications. Whilst the runtime memory usage of this representation is currently dwarfed by the runtime memory usage of the grid and polygonal mesh it is rasterised to, this would also allow for smaller file sizes if support is added for exporting to a lossless file format.

Potential usability improvements

In addition to improvements to application performance, the following changes would be expected to improve the application's usability:

- Allowing custom brushes to be defined, e.g. from existing SDF-based models, polygonal models, or bitmaps. This would open up the possibility for certain parts of an object to be sculpted more efficiently.
- Allowing for models to be saved and loaded in their lossless SDF-based representation, as opposed to exclusively using polygonal meshes for import and export. This would simply require serialising and deserialising the internal operation stack to and from a file.
- Adding more brushes that manipulate matter in ways that don't simply add or destroy it. A tool for proportionally shifting points of the signed distance field (intended to be equivalent to ZBrush and Blender's "Grab" brushes) was investigated and partially implemented for the final product, but is presently unreliable. This in particular would allow for the proportions of a model to be more efficiently adjustable, or become a means of adding detail to a model itself.
- Further improvements to the depth perception problem outside of VR, e.g. adopting something similar to ZBrush where something is drawn alongside the normal of an area being modified. This would be particularly useful for aforementioned brushes that shift and warp the distance field as opposed to adding and removing mass.

- Further refinements to the control scheme, ideally in response to some kind of solicitation of user feedback.
- The import process for existing mesh files could be less crude - methods exist to represent polygonal meshes exactly as signed distance functions, as opposed to the current conversion approach which simply maps vertices to spheres. This would allow for imported meshes to more closely match their original appearance.
- Adding the ability to undo and redo operations. This would be trivial to implement, given that the model is stored internally as a single stack of operations, requiring only the last item in that stack to be removed to undo an operation.
- Allowing colors and materials to be painted on the surface of the model, removing the need to switch applications for texturing. Similar to how the shape of the model is currently represented, colors could also be represented losslessly as operations on a signed distance field, and be rasterised either to a texture or vertex colours. An advantage of such an approach would be that even extreme changes in model topology would not render texture information invalid.

Conclusion

Signed distance functions, and by extension other implicit methods of representing 3D shapes, have thus been demonstrated to form a potentially compelling basis for 3D sculpting applications. The use of signed distance functions in the prototype application enables a paradigm of creating and removing mass at arbitrary points in space that would be difficult to replicate with traditional polygonal representations that are bound by the constraints of topology. This allows for rapid freeform sculpting of 3D objects without the user needing to think about topology or coordinates, with acceptable performance on modern consumer desktop hardware.

Nevertheless, this approach raises unique user experience questions such as to how to handle depth perception when creating matter at arbitrary points in space. The need

to translate the signed distance field to some immediately viewable representation in real time, in this case a polygonal mesh, also introduces some unique performance characteristics that must be accounted for to deliver a sufficiently smooth user experience. Whilst the prototype has demonstrated the viability of such an approach, there are many areas where room for improvement has been identified. Future research is encouraged to improve performance and user experience for sculpting applications based on implicit surfaces, as well as to discover other benefits non-polygonal representations may enable.

References

Angles, B., Tarini, M., Wyvill, B., Barthe, L., & Tagliasacchi, A. (2017). Sketch-based implicit blending. *ACM Transactions on Graphics*, *36*, 1–13. https://doi.org/10.1145/3130800.3130825

Berger, M., Tagliasacchi, A., Seversky, L. M., Alliez, P., Guennebaud, G., Levine, J. A., Sharf, A., & Silva, C. T. (2016). A survey of surface reconstruction from point clouds. *Computer Graphics Forum*, *36*, 301–329. https://doi.org/10.1111/cgf.12802

Chen, S., Xu, R., Xu, J., Xin, S., Tu, C., Yang, C., & Lü, L. (2023). QuickCSGModeling: Quick CSG operations based on fusing signed distance fields for VR modeling. *ACM Transactions on Multimedia Computing, Communications, and Applications*. https://doi.org/10.1145/3599729

Deng, Y., Ni, Y., Li, Z., Mu, S., & Zhang, W. (2018). Toward real-time ray tracing. *ACM Computing Surveys*, *50*, 1–41. https://doi.org/10.1145/3104067

Developers, G. for. (2018). *Asynchronous reprojection | google VR*. Google for Developers; Google for Developers. https://developers.google.com/vr/discover/async-reprojection

Du, H., & Qin, H. (2000). Direct manipulation and interactive sculpting of PDE surfaces. *Computer Graphics Forum*, *19*, 261–270. https://doi.org/10.1111/1467-8659.00418

ephtracy. (2021). *MagicaVoxel*. ephtracy.github.io. https://ephtracy.github.io/index.html?page=magicacsg

Evans, A. (2015). *Learning from failure: A survey of promising, unconventional and mostly abandoned renderers for "dreams PS4," a geometrically dense, painterly UGC game*. https://advances.realtimerendering.com/s2015/AlexEvans_SIGGRAPH-2015-sml.pdf

FAD. (2022). *Implicit surface renderer*. www.shadertoy.com. https://www.shadertoy.com/view/7tKfz1

Ferley, E., Cani, M.-P., & Gascuel, J.-D. (2000). Practical volumetric sculpting. *The Visual Computer*, *16*, 469–480. https://doi.org/10.1007/pl00007216

Ii, R. (2004). *Directly rendering non-polygonal objects on graphics hardware using vertex and fragment programs.* University of North Carolina at Chapel Hill. https://www.cs.unc.edu/~taylorr/raycast_fragment/raycast_fragment.9b.pdf

Nielsen, J. (2019). *Response time limits: Article by jakob nielsen.* Nielsen Norman Group. https://www.nngroup.com/articles/response-times-3-important-limits/

Pauly, M. V. (2003). Point primitives for interactive modeling and processing of 3D geometry. *Selected Readings in Vision and Graphics*, *23*. https://doi.org/10.3929/ethz-a-004612876

Quilez, I. (n.d.). *Inigo quilez.* iquilezles.org. https://iquilezles.org/articles/distfunctions/

Reiner, T., Mückl, G., & Dachsbacher, C. (2011). Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions. *Computers & Graphics*, *35*, 596–603. https://doi.org/10.1016/j.cag.2011.03.010

Sharp, N. J. H., & Jacobson, A. (2022). Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis. *arXiv (Cornell University).* https://doi.org/10.48550/arxiv.2202.02444

Sigg, C. (2006). *Representation and rendering of implicit surfaces.* https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ac68f258020f1af9dd711846a9e98721c6bcd0d7

Söderlund, H. (2021). *Hardware-accelerated ray tracing of implicit surfaces a study of real-time editing and rendering of implicit surfaces.* https://bth.diva-portal.org/smash/get/diva2:1571860/FULLTEXT02.pdf

Takikawa, T., Litalien, J., Yin, K., Kreis, K., Loop, C., Nowrouzezahrai, D., Jacobson, A., McGuire, M., & Fidler, S. (2021). Neural geometric level of detail: Real-time rendering with implicit 3D shapes. *Computer Vision and Pattern Recognition.* https://doi.org/10.1109/cvpr46437.2021.01120

Vaishnavi, V., & Kuechler, B. (2015). *Design science research in information systems.* http://desrist.org/desrist/content/design-science-research-in-information-systems.pdf

YVT. (2016). *Implicit surface rendering.* www.shadertoy.com. https://www.shadertoy.com/view/XsdXzN

Appendix

Communication log

Date	Event	Topic	Outcome
08/03/23	Zoom meeting	Defining research topic and question	Realized chosen topic was difficult to determine a concrete research question for, decided to start searching for different topics in the same domain
14/03/23	Email	Proposing a new research topic and question	Resources provided for new topic
23/03/23	Email	Safety & ethics forms	Safety and ethics forms signed
05/04/23	Zoom meeting	Methods and tooling for approaching problem	Provided more resources and prior research on the topic
18/05/23	Email	More detailed research methodology and details on artifacts	Set up an additional meeting to further discuss methodology and artifacts
06/05/23	Email	Setting up regular Zoom meetings	Scheduled fortnightly Zoom meetings

Date	Event	Topic	Outcome
21/06/23	Zoom meeting	Progress update: setting up and compiling tools	Decided to read up more on GLFW, OpenVDB, and Imgui. Also considering Godot as a possible middle ground.
26/07/23	Email + Zoom meeting	Progress update	Raised resources used for SDF functions and existing software, and raised questions about multiresolution grids
02/08/23	Zoom meeting	Progress update	Try adding mesh import feature and smoothing model
23/08/23	Email + Zoom meeting	Progress update	Showed progress on smoothed blending functions, provided repository link, considered going down route of freeform sculpting
02/09/23	Zoom meeting	Progress update	Formally started thesis paper, raised further questions about depth perception

Date	Event	Topic	Outcome
06/09/23	Zoom meeting	Progress update	Consider allowing imported meshes to be used as brushes, and support for serializing to both mesh and lossless formats
20/09/23	Zoom meeting	Progress update	Try implementing custom brushes based on bitmaps, consider using ZBrush's approach to cursor depth perception
23/10/23	Zoom meeting	Progress update	Think about interface for multiple brushes, consider adding copy/paste functionality. Advice given for thesis structure.
27/10/23	Zoom meeting	Progress update	Sent first draft of thesis paper
01/11/23	Zoom meeting	Progress update	Addressed feedback for lighting setup
13/11/23	Zoom meeting	Thesis paper feedback	Addressed shared feedback