Echocardiography Guidance and Training with Mixed Reality

Edvart G. Bjerke

March 2025

1 Introduction

The aim of this thesis is to implement a guidance and training system in augmented reality. The system is targeted at trainees, students and other unskilled practitioners. The goal is to provide a tool that is helpful both in learning about how to perform echocardiographic examinations and in actual examinations.

A pre-trained deep neural network- based model will be used to generate guidance signals for the user, and these signals will be integrated into the augmented reality system. Additionally, virtual reference model placed in the virtual environment allows the user to explore cardiac structures and familiarize themselves with standard views.

The reference models are based on cardiac MRI data. Since the project is focused on echocardiography, it's essential that the reference model also includes ultrasound data. This data should be spatially complete such that the user can view scan planes from arbitrary poses in a continuous manner. To maintain coherence, all representations (raw MRI, solid model and ultrasound) are obtained from the same patient. This ensures that context is not lost when switching between representations.

Obtaining dense 4D ultrasound data that coheres with the MRI data from manual scanning is impractical. Instead, echo data is simulated based on the solid model manually generated by contouring the 4D MRI data.

2 Echocardiography

Echocardiography examinations provide information about the patient's heart used to diagnose various cardiovascular disorders. Today, echocardiography is used extensively, as the procedure is low-cost and non-invasive [1].

In echocardiography, ultrasound signals are transmitted by a probe. The signal propagates through the medium of the patient and echoes are recorded by the probe. These echoes are processed by a set of signal processing algorithms commonly referred to as "beamforming". The result of beamforming is a greyscale image representing anatomical structures.

3 Computer Graphics

Computer graphics (CG) is the study of digitally generating images, videos, and other types of graphical content. The type of visual content generated can broadly be categorized into photorealistic and non-photorealistic CG. The applications of CG techniques are broad- including scientific visualizations, video games, special effects and other types of informative or artistic content.

Algorithms in computer graphics typically operate on objects of a three-dimensional (3D) *scene*. The scene is a high-level description of the virtual environment used to synthesize the visual content. The objects in the scene may represent things like solid bodies, cameras, light sources or arbitrary volumetric data.

3.1 Rendering

Rendering algorithms generate digital images, given a set of objects from the scene. These algorithms are often physically-based, where *photorealistic* techniques simulate real light phenomena to produce visually realistic results.

Non-photorealistic rendering techniques are used when photorealism is n ot desired- whether for aesthetic purposes or when the objects represent something invisible to the human eye.

3.2 The Graphics Pipeline

The graphics pipeline is a high-level description of the sequence of processes used to generate an image from the scene description. The graphics pipeline is not rigidly defined, and different architectures are used to render different tyepes of data or to achieve results of a particular aesthetic.

In real-time computer graphics, the graphics pipeline can generally be viewed as three parts- the application, geometry processing and rasterization [2].

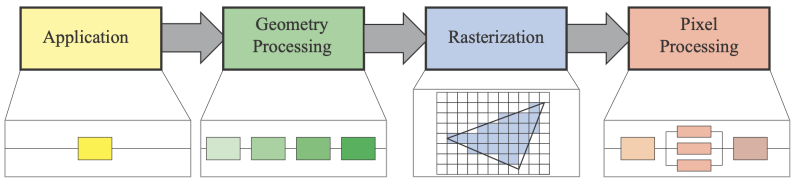

Figure 1: The graphics pipeline [2]

The application part of the pipeline generally runs on the CPU, and is concerned with defining the structure of the scene, setting up rendering commands and other pre-processing procedures. The application defines a set of geometry *primitives* which are sent further down the pipeline. These primitives are geometrical shapes like triangles and lines, defined by coordinates and indices

defining the surface of the models to be processed and rendered by the GPU. The data strutures representing these primitives are typically defined in a local coordinate system referred to as *object space*, and may include additional information such as surface normals and texture coordinates. The data required in this stage depends on the algorithms employed in the *pixel processing* stage of the pipeline.

The geometry processing stage is executed on the GPU, where geometrical operations like morphing and subdividing (tesselation) may be performed. An essential part of the geometry processing stage is the *vertex shader*, which is a programmable procedure which processes each 3D vertex (position) of the given mesh.

In a typical rendering pipeline, the vertex shader transforms the vertex coordinates from *object space* to *screen space* through perspective projection, such that objects are correctly rendered to the screen according to the *pinhole camera model*.

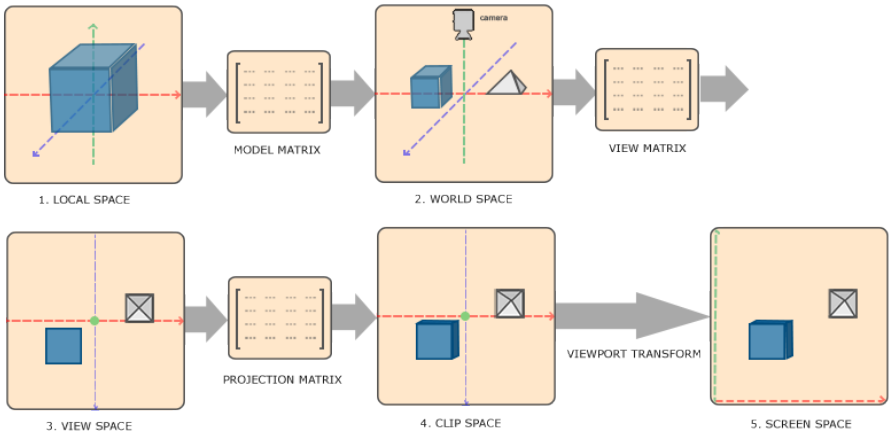

Figure 2: Representation of geometry processing in a standard vertex shader using perspective projection[3].

The rasterization part of the pipeline uses the screen space coordinates computed by the vertex shader to determine the set of pixels covered by the projection of the primitive.

Finally, the colors of the rasterized pixels are determined in the pixel processing stage. The GPU program defining the algorithm used for shading pixels is referred to as the *fragment shader*. The fragment shader is executed in parallell, for each rasterized pixel. With modern GPUs and graphics frameworks, fragment shaders are highly programmable, and different shaders may be used to render different objects.

3.3 Volume Rendering

The traditional raster-based rendering technique is not well-suited for rendering phenomena that are volumetric in nature. Special rendering techniques have been developed to render physically occuring phenomena like smoke, clouds or fog. These techniques may also be used to visualize three-dimensional scientific data, like that captured by fluid simulation or medical imaging.

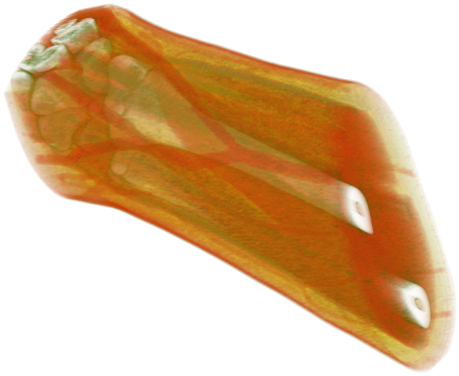

Figure 3: Volume rendering of a CT scan of a forearm

3.3.1 Volume Data Representation

3.3.2 Cut Planes

A simple way to render volumetric data is reduce the problem to two dimensions, by only rendering a single *slice* of the data at once. This can be achieved by rendering a plane placed such that it intersects with the bounding volume of the data set. In the fragment shader, the world space position of each fragment is used to sample the volumetric data set. The sampled value can then be used to determine the final color of the pixel via some transfer function.

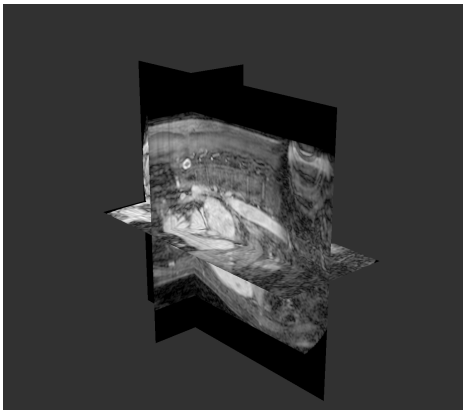

Figure 4: Volume rendering of an MRI dataset using three cut planes.

This method is easy to implement and interpret, requiring only a simple modification to the fragment shader.

3.3.3 Volume Ray Casting

Volume Ray Casting is a rendering technique used to directly render volumetric scalar fields. Intuitively, a ray is projected from each pixel into the volume, and the scalar field is iteratively sampled along each ray to compute the pixel values. This algorithm approximates the volume rendering rendering integral by computing the Discrete Volume Rendering Integral (DVRI)

$$I(D) = \sum_{i=0}^{n} C_i A_i \prod_{j=i+1}^{n} (1 - A_j)$$

Where I is the accumulated intensity for a given pixel with total ray depth D. C_i and A_i represent the intensities and opacities sampled along the ray, respectively. This can be extended to color images by performing the same algorithm for each channel in the color image (RGB).

If ray-casting is performed front-to-back, the algorithm takes the following form [4]

$$C_{dst} \leftarrow C_{dst} + (1 - \alpha_{dst}) C_{src}$$

$$\alpha_{dst} \leftarrow \alpha_{dst} + (1 - \alpha_{dst}) \alpha_{src}$$

Here, the *destination* color C_{dst} and opacity A_{dst} are computed by *compositing* the result from the previous iteration with the *source* values, sampled at the current ray segment. To sample arbitrary positions with the volume, trilinear interpolation is used. A shading model may also be applied during ray traversal, and this typically requires computing the gradients of the scalar field.

Transfer functions are used to map data values from the 3D scalar field to colors and opacities. The choice of transfer function depends on the desired visualization and may require expert insight into the nature of the underlying data. The steps of the volume ray casting algorithm are demonstrated in figure 5.

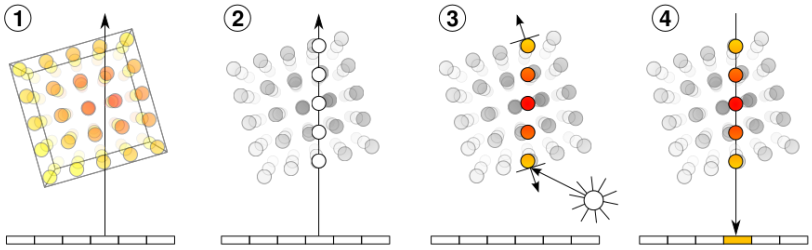

Figure 5: Volume ray casting. The generated ray for a given fragment is shown in (1), passing through the discrete scalar field. In (2), data values are sampled along the ray. Then, a transfer function and shading model are applied at each sample (3) to compute color and opacity values. Finally, the pixel value is computed through compositing (4) [5].

This algorithm can be implemented on the GPU by conforming to the traditional rasterization pipeline by using *proxy geometries*. The proxy geometries are placed such that the rasterized pixels cover the projection of the scalar field in view space. Thus, the ray casting algorithm can be implemented directly in the fragment shader used for these geometries.

3.3.4 Transparency

(Write about ray termination and rendering order)

References

[1] S. Omerovic and A. Jain, *Echocardiogram*. Treasure Island (FL): StatPearls Publishing, updated 2023 jul 24 ed., 2025.

[2] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*. CRC Press, 4 ed., 2019. [Originally published in 2008].

[3] J. de Vries, "Figure from learnopengl: [vertex transforms]." Online, 2025. Licensed under CC BY-NC 4.0. Accessed: 28-Mar-2025.

[4] J. Krüger and R. Westermann, "GPU-based interactive visualization techniques," in *Proceedings of IEEE Visualization 2006*, pp. 13–20, IEEE, 2006.

[5] F. Hofmann, "Volume ray casting," 2011. CC BY-SA 3.0, via Wikimedia Commons.