2016

Massive Model Visualization: A Practical Solution

Jeremy S. Bennett

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Computer Engineering Commons, Computer Sciences Commons, and the Mechanical Engineering Commons

Recommended Citation

https://lib.dr.iastate.edu/etd/15663

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Massive Model Visualization: A practical solution

by

Jeremy Scott Bennett

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Co-Majors: Human Computer Interaction; Computer Engineering

Program of Study Committee:
James H. Oliver, Co-Major Professor
Joseph Zambreno, Co-Major Professor
Yan-Bin Jia
Rafael Radkowski
Chris Harding

Iowa State University
Ames, Iowa
2016

Copyright © Jeremy Scott Bennett, 2016. All rights reserved.
# TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................ v

LIST OF TABLES ........................................................................................................ vii

Acknowledgements .................................................................................................. viii

Abstract ................................................................................................................... ix

CHAPTER 1. INTRODUCTION ............................................................................... 1
  1.1 Motivation ........................................................................................................... 1
  1.2 Summary of Research ......................................................................................... 4
    1.2.1 GPU Based Depth Buffer Reprojection ....................................................... 4
    1.2.2 Batch Query ................................................................................................ 5
    1.2.3 User Study Perceived Quality ..................................................................... 5
  1.3 Dissertation Organization .................................................................................... 6

CHAPTER 2. RELATED WORK ........................................................................ 7
  2.1 MMR .................................................................................................................... 7
  2.2 Far Voxels ........................................................................................................... 9
  2.3 Interviews3D ....................................................................................................... 11
  2.4 CPU Based Culling ............................................................................................ 13
  2.5 GPU Based Culling ............................................................................................ 14
  2.6 Massive Model Visualization: An Investigation into Spatial Hierarchies ........ 15
    2.6.1 Spatial Hierarchy .......................................................................................... 16
    2.6.2 Visibility Determination Algorithm ............................................................ 18
    2.6.3 Results ......................................................................................................... 20

CHAPTER 3. SYSTEM ARCHITECTURE .............................................. 22
  3.1 Spatial Hierarchy Design ................................................................................... 22
  3.2 Visibility Determination Algorithm ................................................................... 23
3.2.1 Render List Generation ................................................................. 24
3.2.2 Render List Render ................................................................. 25
3.3 Baseline Performance ................................................................. 26

CHAPTER 4. GPU DEPTH REPROJECTION .............................................. 28
4.1 Alternative Approaches ................................................................. 29
4.2 GPU Depth Reprojection Design .................................................. 31

CHAPTER 5. GPU BATCH QUERY ............................................................. 32
5.1 GPU Batch Query Design .............................................................. 33
5.1.1 Shader Buffer Write ................................................................. 33
5.1.2 Multi-Draw Indirect ................................................................. 35
5.1.3 System Integration ................................................................. 36

CHAPTER 6. USER STUDY DISOCCLUSION ARTIFACTS .................................. 38
6.1 Study Design ........................................................................... 38
6.2 Study Setup ........................................................................... 38

CHAPTER 7. RESULTS ............................................................................. 42
7.1 Accuracy .................................................................................. 42
7.1.1 Manual Inspection ................................................................. 43
7.1.2 Complete Initial Render ....................................................... 45
7.1.3 360 Degree Rotation ........................................................... 47
7.1.4 Depth Buffer ......................................................................... 49
7.2 Performance ............................................................................... 50
7.2.1 360 Degree Rotation ........................................................... 50
7.2.2 User Study Path ..................................................................... 55
7.3 Perceived Quality ........................................................................ 61
7.3.1 Survey Demographics .......................................................... 61
7.3.2 Survey Response ................................................................. 62
7.3.3 Survey Rank ......................................................................... 67
7.3.4 Survey Correlation ............................................................... 68
7.3.5 Discussion ............................................................................ 69
CHAPTER 8. CONCLUSION AND FUTURE WORK................................................................. 70

BIBLIOGRAPHY ............................................................................................................. 72
LIST OF FIGURES

Figure 1: Vertex count in visualization papers [1] ............................................................. 1
Figure 2: Performance of traditional LMV approach verses a MMV approach [2] ............ 2
Figure 3: Basic MMV System Layout............................................................................. 3
Figure 4: MMR viewpoint cell (right) and algorithm (left) [3] ......................................... 8
Figure 5: Far Voxels [11] ............................................................................................ 9
Figure 6: Boeing 777 rendered in real time by Interviews3D [6] ..................................... 11
Figure 7: Single Precession Floating Point Performance for CPU and GPU .................. 14
Figure 8: Octree spatial hierarchy............................................................................... 16
Figure 9: Multithreaded octree subdivision system..................................................... 17
Figure 10: Octree serialization and extraction time when run on multiple threads ......... 18
Figure 11: Visibility Determination Algorithm.......................................................... 19
Figure 12: Comparison of MMV and LMV Rendering Frame Rates ......................... 20
Figure 13: Visibility Determination Algorithm.......................................................... 23
Figure 14: LMV verses MMV performance under current system architecture ............. 26
Figure 15: Percentage of CPU Time in each Stage for Generation 1 Algorithm ............. 27
Figure 16: Percentage of GPU Time Each Stage for Generation 1 Algorithm ............... 27
Figure 17: Resulting color buffer for nominal, without quad, and with quad ............... 30
Figure 18: GPU Depth Reprojection......................................................................... 31
Figure 19: Spatial Hierarchy Buffer Layout.............................................................. 34
Figure 20: glMultiDrawElementIndirect Specification............................................. 35
Figure 21: MEDI Buffer Layout............................................................................... 36
Figure 22: Overview Generation 2 Strategy ............................................................. 37
Figure 23: Approximate User Study Scene ............................................................... 40
Figure 24: Approximate Path of User Study Animation with Potential Selection Events ... 41
Figure 25: Generation 1 Inside of Geometry ............................................................. 44
Figure 26: Generation 2 Oscillation ....................................................................... 44
Figure 27: Depth Reprojection Plane Z-Fighting....................................................... 45
Figure 28: Visible Occurrences (Left) and Rendered Triangles (Right) while Rotating 360 .... 48
Figure 29: Missing Occurrences (Left) and Extra Occurrences (Right) while Rotating 360

Figure 30: Reprojected depth buffer blended over the original mesh

Figure 31: Speed Up When Rendering 360 Degrees Relative to Generation 1

Figure 32: Per Frame Performance for the Cotton Picker (Left) and Air Craft (Right)

Figure 33: Per Frame Render Items for the Cotton Picker (Left) and Air Craft (Right)

Figure 34: Rate of Render Item Change for the Cotton Picker (Left) and Air Craft (Right)

Figure 35: CPU Performance of Cotton Picker for 360 Degree Rotation

Figure 36: CPU of Air Craft for 360 Degree Rotation

Figure 37: User Study Time per Path Key Frame: Overall

Figure 38: User Study Time per Path Key Frame Event Normalized

Figure 39: Number of Frames per Path Frame During User Study

Figure 40: Redraw Time of User Study Path

Figure 41: Number of Render Items Produced by each Algorithm during User Study

Figure 42: User Study Path Performance Reprojection 1

Figure 43: User Study Path Performance for Reprojection 2

Figure 44: User Study Demographic Information

Figure 45: Question 1 Results (Left) Mean and Confidence Interval (Right)

Figure 46: Question 2 Results (Left) Mean and Confidence Interval (Right)

Figure 47: Question 3 Results (Left) Mean and Confidence Interval (Right)

Figure 48: Question 4 Results (Left) Mean and Confidence Interval (Right)

Figure 49: Question 5 Results (Left) Mean and Confidence Interval (Right)

Figure 50: Question 6 Results (Left) Mean and Confidence Interval (Right)

Figure 51: Question 7 Results (Left) Mean and Confidence Interval (Right)

Figure 52: Question 8 Results (Left) Mean and Confidence Interval (Right)

Figure 53: Question 9 Results (Left) Mean and Confidence Interval (Right)

Figure 54: User Study Survey Results Based Upon Individual Ranking
LIST OF TABLES

Table 1: Post Animation Survey ........................................................................................................... 39
Table 2: Participant Demographics Survey .......................................................................................... 40
Table 3: Visibility Determination Algorithms ...................................................................................... 41
Table 4: Approaches Used for Testing .................................................................................................. 42
Table 5: Cotton Picker Accuracy ......................................................................................................... 46
Table 6: Air Craft Accuracy .................................................................................................................. 47
Table 7: Time to Complete User Study Animation .............................................................................. 55
Table 8: Correlations within Survey Results ......................................................................................... 69
ACKNOWLEDGEMENTS

Without the assistance of colleagues, advisors, friends, and most importantly, family, the present research would not have been possible. I would like to express gratitude to my major professors and committee members: Dr. James Oliver who has served as both friend and mentor for more years than I wish to count; Dr. Joseph Zambreno who went above and beyond expectations to insure I kept one foot in front of the other; Dr. Yan-Bin Jia and Dr. Chris Harding for their patience and understanding; and finally, Dr. Rafael Radkowski for agreeing to join my committee at the last moment.

I would like to thank the many colleagues at Siemens PLM Software who have supported my efforts and served as sounding boards for the many technical questions I faced: Dr. Michael Carter for serving as both a mentor and a friend; Dr. Adam Faeth who started this journey with me and has encouraged me along the way; and Tony Deluca, Kevin Glossner, and Brad Halls for helping me navigate various Siemens’ processes and policies.

I would especially like to thank my friends and family without whose support this would not have been possible: my wife Audra for her infinite patience and love; my son, Waylen, for the infinite distractions that helped keep me sane; and lastly, my son, Cassius, for giving me reason to stay up late and get the work done.
ABSTRACT

The ever-increasingly complex designs emanating from various companies are leading to a data explosion that is far outstripping the growth in computing processing power. The traditional large model visualization approaches used for rendering these data sets are quickly becoming insufficient, thus leading to a greater adoption of the new massive model visualization approaches designed to handle these arbitrarily sized data sets. Most new approaches utilize GPU occlusion queries that limit the data needed for loading and rendering to only those which can potentially contribute to the final image. By doing so, these approaches introduce disocclusion artifacts that often reduce the quality of the resulting visualization as a camera is maneuvered through the scene. The present research will demonstrate that shader based depth reprojection and OpenGL atomic writes not only increase the performance of an existing system based upon OpenGL occlusion queries, but also reduce the amount of perceived disocclusion artifacts.
CHAPTER 1. INTRODUCTION

1.1 Motivation

The digital era has promoted the adoption of computer systems throughout an entire product development lifecycle. Companies are now more capable than ever of breaking new ground with each successive generation of their products, resulting in ever increasing complex designs. This, in turn, is leading to a data explosion far outpacing increases in the processing power of computer systems used to build them, as shown in Figure 1. The current generation of CAD and Visualization software is no longer capable of rendering the current generation of airplanes and ships in their entirety.

![Figure 1: Vertex count in visualization papers [1]](image)

The current generation of software is designed around Large Model Visualization (LMV) technologies. These technologies work by traversing a product structure (usually represented as a scene graph) and by using various techniques such as view frustum culling, size culling, occlusion culling, and level of detail representation to limit the number of polygons rendered when viewing the scene. The problem with this approach is that it is linear in nature, and therefore computational cost grows at the same rate as incremental data size.
"Massive Model Visualization" (MMV) is the term used to encompass new technologies designed to handle this problem. The key principle of MMV is that the number of polygons that can potentially contribute to a rendered image from a given viewpoint is limited by the total number of pixels available in the image, not by the total number of available polygons. A 1080p high definition screen has a resolution of 1920 by 1080, or just over 2 million pixels. If one were to render a relatively large model of 200 million triangles, only 2 percent of those triangles could possibly contribute to the final image. MMV technologies are about creating a system that is bound by screen space, not data size, which is definitely not the case with systems based purely on LMV technologies, as shown in Figure 2.

![Figure 2: Performance of traditional LMV approach verses a MMV approach [2]](image)

There are several components included in a MMV system that makes this possible, as shown in Figure 3. As part of a preprocess operation, the product structure (usually in the form of a scene graph) is subdivided into spatial hierarchy and data cache. The data cache can contain anything from occurrences, polygons, or voxels, depending upon the level of subdivision deemed necessary. A strategy pass is executed during render over the spatial hierarchy in order to construct a visibility list of all data expected to contribute to the current frame. This information is then fed into the
renderer to generate the final image, the loader to ensure that any required data is resident in the geometric cache, and the reaper to ensure that recent unused data is unloaded from memory. The operations within the render block can be executed in parallel. The strategy can generate the visibility data for the next frame while the renderer is still rendering the current frame, and the loader and reaper can run in a constant cycle, executing data load and unloads as necessary.

For a MMV system to be successful, all components must be in place: each plays a critical role in handling extremely large datasets. The spatial hierarchy generated by the partitioner must provide enough spatial coherence between the cells for the strategy to cull large batches of geometry efficiently. The partitioner must also ensure that the data contained within the cells are sufficiently coarse to minimize the amount of noncontributing geometry used. One of the biggest challenges with extremely large datasets is that they contain vastly more geometric information than the main memory can manage. Render components must work together to manage the amount of data resident at any given point in time. The loader is responsible for loading data; it needs to be agile enough to ensure that data is quickly available when marked as needed. Often

![Figure 3: Basic MMV System Layout](image)
predictive algorithms are used by the loader to prefetch likely visible data, minimizing any potential lag. The reaper is responsible for both detecting and unloading data when it is no longer necessary, and for determining the best candidates for unloading when memory approaches maximum threshold. The strategy’s primary responsibility is to construct a list of visible occurrences. This is accomplished by executing advanced culling techniques against the spatial hierarchy to determine the set of occurrences most likely to contribute to the final image. Occlusion tests serve as the primary culling technique for most approaches.

Culling techniques are not without limitations: they inherently introduce disocclusion artifacts. Disocclusion artifacts occur whenever a visible shape is not rendered for one or more frames while visible, thus causing a perceivable popping effect when finally rendered. This behavior often results from the visibility determination algorithm’s inability to keep up with the visibility state changes occurring as the camera is moved through the scene, or if the loader fails to load data before it is needed.

1.2 Summary of Research

GPU based occlusion tests have been shown to be an effective tool for improving rendering performance in both industry and games. The present dissertation presents novel improvements to a GPU based occlusion strategy for improved performance and reduced disocclusion artifacts. It also presents a simple user study for evaluating how these improvements affect perceived quality.

1.2.1 GPU Based Depth Buffer Reprojection

GPU based occlusion tests require that the z-buffer be prepopulated with potential occluders depth values. Most approaches accomplish this by rendering either a potential occluder list or the existing render list into the depth buffer. On large models this can involve rendering millions of triangles, far more than the number of pixels on the screen and at significant computational cost. A commonly used principle
with Massive Model Visualization techniques is frame-to-frame coherence, or the notion that the visibility state of occurrences will not change significantly between frames. It can thus be extrapolated that the zbuffer used for occlusion culling will not alter significantly. The present research shows how the depth buffer from a previous frame can be reprojected into the current viewpoint to approximate the current z-buffer using a textured depth mesh and simple shaders at little cost.

1.2.2 Batch Query

GPU based occlusion strategies frequently rely on GL Occlusion Queries to determine the visibility of entities. While these provide an easy means for using the GPU to determine if a given set of primitives will contribute to the final image, they severely limit the amount of achievable parallelism on the GPU, as entities have to be queried one at a time. While newer GL extensions have improved upon this functionality by allowing multiple results to be retrieved simultaneously, it still comes at significant cost.

GPU based Atomic Increment has been shown to be a viable alternative to GPU Occlusion Queries, allowing the visibility of all entities of interest to be obtained with a single draw call, thereby significantly increasing GPU parallelism. The present research shows how this can be effectively combined with spatial hierarchy to increase its scalability to arbitrarily large data sets.

1.2.3 User Study Perceived Quality

A common side effect of occlusion based culling techniques is the popping in and out geometry as it transitions between visible and invisible states. These disocclusion artifacts can have a severe impact on the perceived quality of the rendered image. This research also presents an evaluation of how novel improvements affect perceived quality through the execution of a simple user study.
1.3 Dissertation Organization

The remaining chapters of this dissertation will provide the necessary background and implementation details for the executed research. Chapter 2 will present the current research trends in Massive Model Visualization. Chapter 3 will provide the high-level system architecture enhanced as part of this research. Chapter 4 will provide a detailed description of how frame-to-frame coherence can be applied to depth buffer generation. Chapter 5 will show how shader buffer write and multi draw instance indirect can be combined with existing spatial hierarchy to form a substantially better query algorithm capable of querying the entire spatial tree in a single call. Chapter 6 will present the user study used to evaluate the effects of disocclusion artifacts on perceived quality. Chapter 7 will provide detailed results of novel improvements and how they compare to the original algorithm. Chapter 8 will provide conclusions drawn from this research and suggestions for future study.
CHAPTER 2. RELATED WORK

Over the last several years there has been a wide array of approaches to achieving interactive framerates on arbitrary large models. While the scope and complexity of systems vary greatly, they are all built on the foundation of using occlusion queries to significantly reduce the amount of data required for rendering. Several systems and approaches will be introduced below to further describe the scope of the problem.

2.1 MMR

MMR, Massive Model Rendering, was designed and implemented as part of the Walkthru Project at the University of North Carolina in Chapel Hill. Like Interviews 3D, its developers realized that CAD model sizes are increasing at a much faster rate than the rendering capability of commodity hardware. A new approach was needed to render these Massive Models containing more than 1 million primitives at interactive framerates of 20 frames per second or more. MMR was built from the ground up to be an extensible platform for enabling further research into arbitrarily large models that can easily scale with model size. Its design allows researchers to interchange various techniques while processing these massive models.

The basic strategy employed by MMR is to avoid rendering any geometry the user will not see, and it is facilitated by a two-part technique. The first part eliminates any geometry that is far away from the viewer by using an image replacement technique. The second part optimizes the rendering of nearby geometry through common acceleration techniques such as occlusion culling and level-of-detail. Input data must be pre-processed to make use of these techniques.
The image replacement technique replaces faraway objects with textured depth meshes. The model space is sub-divided into a set of viewpoint cells using a view emphasis function to determine which objects are considered far from the viewpoint. The view emphasis function is a user-defined function that is unique for each model, returning a scalar importance measure for any point in the model space that allows for the area importance to be easily determined. Each viewpoint cell is given a large cull box that may overlap with neighboring cell cull boxes. Objects outside a cell’s cull box are considered far from the viewer when standing inside the cell. A set of depth maps are created for each viewpoint cell and contains resulting color and depth values for the geometry outside the cull box when looking from the center of the cell through each of its sides. This data is used to generate a simplified texture depth mesh for each side that can serve as a necessary stand-in using algorithms from Darsa and Sillion for heavy reductions in complexity, and Garland and Heckbert for fine tuning reductions [4, 5, 6].

Each object is processed so that large objects intersecting multiple viewpoint cells can be split into sets of smaller objects, thereby facilitating the use of techniques required to render near-geometry. Further, alternate representations are created for each object using a minor variation of a method proposed by Erikson and Manocha [7]. Last, potential occluder objects are calculated for each cell.

MMR uses a four-process approach to handle renderings of arbitrarily large models. The main process is devoted to rendering and handling operations occurring between phases. Two sub-processes are devoted to handling culling, one of which is
devoted solely to occlusion culling. The last process handles asynchronous I/O and attempts to pre-fetch any texture depth maps or tri-strips needed in subsequent frames [8, 9, 10].

### 2.2 Far Voxels

Far Voxels is a volumetric approach for rendering arbitrarily large models. Realizing that model size is quickly outpacing the graphics capability of most commodity hardware, Gobetti and Marton set out to create a method that will work on a larger subset of models that is better than currently available output-sensitive approaches.

![Figure 5: Far Voxels](image)

Gobetti and Marton’s approach uses a pre-processing step to generate a coarse volume hierarchy whose leaf nodes contain a fixed number of triangles, and interior nodes contain a voxel approximation of underlying nodes. The hierarchy is generated by using the surface heuristic approach, as defined by MacDonald and Booth, to subdivide the model into an axis-aligned BSP tree [8], and the resulting tree is stored in memory coherent order to improve cache locality, similar to one proposed by Harvan [9]. Once complete, the tree is combined with a coarse hierarchical data structure by first removing empty nodes, and then associating sub-trees containing a set number of BSP tree triangles to leaf nodes in the hierarchy. The hierarchy is traversed to finalize each leaf node, and immediately saved out to disk once the data structures have been combined. Leaf nodes are finalized by extracting triangles from the associated BSP-tree,
culling out triangles outside the nodes bounding box, tri-striping resulting triangles, and performing optimizations to increase cache coherency of the resulting data. The interior nodes view-dependent voxels are generated by casting a large random set of rays against the BSP tree at each node to generate a set of pixel samples from approximately all un-occluded directions. These samples are then fitted to a set of shader models to compress their size and allow for more efficient rendering. The input data set is split into chunks containing between 20 and 30 million triangles to take advantage of the massive parallelism that exists within the subdivision approach. These chunks are then farmed out to a networked group of computers that generate a tree for the model’s respective section.

Rendering occurs in breadth-first, front-to-back order on a set of trees. A priority queue is used to sort the current set of nodes to be potentially rendered in a front-to-back order, with the initial set of nodes for each frame being the root nodes of all trees. While there are still nodes in the queue, the front node is removed and processed for rendering. All nodes are initially marked as invisible in the current frame, so during processing the active node can simply be discarded, resulting in it and its whole sub-tree being culled from the current frame. For example, if the active node's bounding box is completely outside the view frustum, it is discarded without going through the occlusion query machinery. The occlusion query machinery handles each node based upon the node’s state. Nodes that were invisible in the previous frame are initially queried against their bounding box. Leaf nodes determined to be below a certain screen coverage threshold, and nodes with children not yet loaded, are queried against their actual render. For all other nodes, their children are pushed directly into the potentially-render queue. The occlusion results are handled once they become available, and nodes whose queries result in zero visible pixels are immediately discarded. All other nodes, and their ancestors, are marked as visible and their children are pushed into the queue. Further, any node whose bounding box was rendered is rendered normally. If the queue should become empty while there are still outstanding
queries, the node associated with the top-most query is handled without waiting on the query to complete.

The loading of nodes is handled through an asynchronous I/O mechanism, and the list of nodes to be loaded is processed at each frame. Fetch requests are issued for as many nodes as can possibly be handled in a given amount of time. Nodes are pre-sorted based upon the estimated voxel size of their parent in order to give a higher priority to nodes that will potentially contribute to a much larger region of the screen [11].

2.3 Interviews3D

One of the most complete systems available for Massive Model Visualization is Interviews3D, supporting navigation, picking, manipulation, animation, and even collision detection. It is described by its creators, 3D Interactive, as a digital mockup system capable of handling data sets with millions, if not billions, of triangles on commodity hardware by using a visibility-guided rendering approach.

Figure 6: Boeing 777 rendered in real time by Interviews3D [6]

Interviews3D was built around a set of pre-defined principles that helped guide and influence its development. The key principles that all systems can utilize include: (a) determine rendering performance by output complexity instead of data size, (b) load and unload data automatically, and allow it to exist in memory only when needed, (c) exploit temporal coherence between frames whenever possible, (d) update rather than rebuild, and (e) utilize current and future technologies whenever possible.
In order to efficiently handle the visualization of arbitrary large models, Interviews 3D uses a preprocessing step to generate an axis-aligned bounding box tree out of input data through a top down approach, resulting in leaf cells containing between 1000 and 8000 triangles. The exact algorithm used for subdivision is not provided, however some key features are explicitly stated. The resulting tree can be selectively updated based upon changes in the original data set, which is significantly more efficient than rebuilding the whole tree. Available memory is taken into account when processing the tree, and if necessary, a multi-pass approach that works only on a subset of the input data is used. Bounding boxes are kept as uniform in size as possible while splitting the side lengths of a cell. Individual triangles are never split or duplicated, meaning there may be some overlap between the bounding boxes of child cells. Subdivision is based upon a minimum number of polygons in a given cell and is calculated for the whole input data set based upon a variety of factors.

The use of spatial hierarchy allows Interviews3D to achieve an interactive framerate through the use of nothing more than an occlusion algorithm and a simple LOD mechanism. The occlusion algorithm is very similar to the one presented in GPU Gems 2, making full use of the GPU capabilities of its time and taking into account the temporal coherence in the geometry rendered from one frame to the next [12]. In order to do this, each cell in the spatial hierarchy is labeled “visible”, “invisible”, or “untested.” Rendering uses these states to carry out a two-pass rendering approach. In the first pass, all shapes marked as “visible” are rendered to prime the depth buffer for occlusion queries. In the second pass, the tree is recursively traversed in a front-to-back, top-down manner using standard view frustum and occlusion tests. When a cell labeled as “invisible” or “untested” is encountered, an occlusion query is immediately executed and the traversal is pruned. Obtaining the results of each occlusion query occurs in parallel to the tree traversal in order to prevent stalls in the graphics pipeline. A post-render tree traversal propagates the visibility state up the graph. A simple LOD mechanism determines the cells that would contribute a pixel or less when rendered to the screen, and renders a single point instead of the cell’s polygons.
Data management occurs asynchronously within Interviews3D and is coupled tightly with the rendering approach. The main objective is to load as many leaf cells as possible into memory, based upon their priority order (respectively, from highest to lowest: visible, invisible but within view frustum, invisible, and finally single-pixel cells). If cells should need to be removed from memory, the cells with lowest priority are removed first, based upon which cell has been in memory the longest [2, 6].

2.4 CPU Based Culling

Coverage buffer culling covers the new class of CPU based occlusion culling techniques that have become popular among major gaming engines. The primary goal of coverage buffer culling is to shift visibility culling back to the CPU to devote all GPU resources toward rendering and achieving ultra-realism at consistently high frame rates. The steady increase in available cores has made it possible to execute complex culling techniques on the CPU with limited to no-impact on other operations. The basic premise of coverage buffer culling is to execute a software rasterisation of simplified objects such as bounding boxes that are then tested against a depth buffer to determine visibility.

FrostBite 2 by dice accomplishes the above by rasterizing a coarse depth buffer using low polygon occluder meshes. The depth buffer can then be used to cull all objects by executing a screen space test [13]. CryENGINE 3 by CryTek reads the previous frame’s depth buffer back from the GPU to the CPU after executing a downscaling pass on the GPU. On consoles such as the Xbox 360 and the PS3 this can be accomplished with only 1 frame of latency, however on a PC this can take up to 4 frames. A coarse rasterisation of objects AABBs and OBBs are used for calculating visibility on a separate CPU thread. The problem with this solution is that there can be a significant difference in the depth buffer from the previous and current view-point resulting in invalid visibility results. The above problem is solved by executing CPU side reprojection, using a point splatting technique on the same thread used to read back the
buffer from the GPU. A 3x3 dilation pass is used in an attempt to stitch any holes that may have opened up as a result of the reprojection [14, 15].

2.5 GPU Based Culling

![Figure 7: Single Precession Floating Point Performance for CPU and GPU](image)

With GPU performance gains far outpacing those of the CPU, there is interest in shifting the entire culling process to the GPU. Tavenrath and Kubisch, researchers from the NVidia SceniX Team, presented a prototype for an advance scene graph rendering pipeline at the GPU Technology Conference (GTC) in 2013. They defined four key points for getting the most out of a render list render. First, the pipeline should share as much geometry as possible. Second, all input parameters should be grouped for fast updating. Third, use MultiDrawIndirect to increase GPU batch size and minimize the number of draw calls. Finally, use the NVidia bindless extension: it provides more flexibility and allows for some of the validation to be bypassed. They also detailed a four-stage culling approach. In the first stage, the previous render list is rendered fully shaded. In the second stage, the entire list of occurrences is tested against the current depth. In the third stage, any newly visible objects are rendered. In the fourth and final stage, update the render list to contain only objects that are visible in this frame. The occlusion test renders the 3 visible sides of the bounding box in the geometry shader, uses the existing depth buffer to perform early z-cull on invisible fragments, and uses texture write to mark which objects are visible [16].
At SIGGRAPH 2015 in Los Angeles, Haar and Aaltonen presented a talk about the GPU Driven Rendering Pipeline of Assassin’s Creed Unity. Designed for the latest generation of hardware, the game saw a massive increase in the amount of geometry to be rendered. For any given scene, rendering individual components would easily result in over 50000 draw calls, almost 10x the desired number. To combat this, they use a mesh cluster rendering approach to improve both batching and culling granularity. The basic principle is to partition all geometry into fixed sized clusters of triangles. This allows glMultiDrawIndexedInstancedIndirect to be used to draw and easily cull an arbitrary number of clusters. Their basic rendering algorithm follows an 8-stage approach with a majority of the work offloaded to the GPU. Stage 1 consisted of a course frustum culling algorithm, stage 2 was responsible for updating all GPU data, and stage 3 batched the draw calls. Starting with stage 4, all work was executed on the GPU beginning with instance culling that utilizes a quad tree based occlusion culler. Stage 5 executed a cluster chunk expansion, while stage 6 executed cluster culling using frustum, occlusion, and backface based techniques. Stage 7 was responsible for generating and compacting the index buffer. Finally, stage 8 executed the scene render. The zbuffer for occlusion testing is generated using a multi-pass process. The first pass renders the best occluders at full resolution, the second pass-down samples the resulting depth buffer, the third pass combines this depth buffer with a low resolution reprojection of the last frame, and the fourth and final pass generates a depth hierarchy for more efficient GPU culling [17].

2.6 Massive Model Visualization: An Investigation into Spatial Hierarchies

In order to develop a complete Massive Mode Visualization solution one must have a solid understanding of the numerous technologies and techniques, and how they are used to accelerate the various sub-components of the system. To this end, a prototype was developed to explore how a spatial hierarchy combined with a visibility determination algorithm can be used to accelerate rendering performance on arbitrarily large models.
2.6.1 Spatial Hierarchy

The spatial hierarchy for the prototype placed a strong emphasis on minimizing the size of individual cells in order to lower the potential L2 cache impact when traversing the tree. The cells were formed into an octree, as shown in Figure 8, because the uniform subdivision allowed for bounding volume information to be calculated dynamically for each cell based upon the path taken to reach the cell, rather than having to be stored. This, combined with other optimizations to limit the amount of state that must be stored, resulted in each cell requiring at most 48 bytes.

As shown in Figure 9, each octree is generated by harvesting the bounding volume information for all occurrences in an assembly extracted from a set of ISO standard Jt files, and subsequently feeding this information into a system for subdividing the tree on multiple threads while minimizing memory usage. The system starts by placing all occurrences on the root cell and then placing it in the queue to be subdivided. The queue is then traversed in order, subdividing any cells that are considered too large to be handled in system memory and subdividing large cells to ensure queue length is sufficient to be handled by multiple threads. Once complete, additional threads are launched and the cell queue is processed as though each entry is the root cell of another spatial hierarchy. The results are then inserted back into the original spatial hierarchy. Cell subdivision begins by first adding occurrences to any child
cell that contains them. Occurrences that intersect multiple child cells are subsequently loaded, and their triangles, along with those pushed down from higher level cells, are added to any child cell that contains them, otherwise they are left on the current cell. Lastly, the cell is serialized to a file and its data are unloaded to conserve memory. Cells are subdivided only if their complexity is greater than a sentinel value of usually 5000 triangles. If the cell does not meet the criteria to be subdivided, its occurrences are loaded, and its triangles are added to the triangle bucket on the cell, serialized and unloaded.

Figure 9: Multithreaded octree subdivision system

The performance of the spatial hierarchy subdivision algorithm was measured by executing the process on the same data set using an increasing number of threads. The algorithm achieved impressive gains for both extraction and serialization when two threads were used, as shown in Figure 10. However, decreases in the extraction time quickly tapered off and no gains were seen beyond five threads, whereas serialization time steadily decreased to as many as twelve threads.
2.6.2 Visibility Determination Algorithm

The algorithm presented in GPU Gems was combined with the algorithm described by 3D Interactive to create a hybrid approach and determine the visibility of cells. Both algorithms focus on rendering only occurrences that can potentially contribute to the final image, while trying to ensure that GPU occlusion queries are interleaved with visible cell renders in order to minimize system stalls when retrieving occlusion query results.

As shown in Figure 11, the algorithm starts by pushing the root cell into a heap that is sorted based upon distance from the viewpoint, and then traversing through the heap by popping the nearest cell. If the cell were considered occluded in the previous frame, an occlusion query is executed by testing the cell’s bounding volume against the depth buffer and pushing the cell number and occlusion id into a FIFO queue of outstanding queries. If the cell were not considered occluded in the previous frame, and it is a leaf, an occlusion query is executed by rendering the cell into both the color and depth buffer and pushing the cell number and occlusion id in the queue of outstanding queries. If the cell were not considered occluded in the previous frame and it contains children, it is rendered into both the color and depth buffers and its children cells are inserted into the heap if they pass a series of CPU based occlusion tests such as view frustum and screen coverage culling. The outstanding occlusion
query queue is checked every 10 cells to determine if the head entry is ready, and if it is, its pixel count is retrieved from the GPU. If the value is less than 5 the cell associated with the entry is marked as occluded. If the value is greater than, or equal to 5 it is marked as not occluded, and if it were previously marked as an occluded cell, it is reprocessed as though it were visible. This process is repeated for all entries that are ready. After both the heap and queue are empty, a depth first traversal is executed over all visible cells, marking any cell whose children are culled as occluded.

![Figure 11: Visibility Determination Algorithm](image)

Testing showed this hybrid approach was able to achieve significant gains over the traditional LMV approach, as seen in Figure 12. While it was not able to achieve the smooth curve that one would expect from an algorithm that was truly bounded only by screen size, it managed to consistently double the frame rate.
2.6.3 Results

The prototype was able to successfully validate the capabilities of MMV technologies and how they could potentially be used to accelerate commercialized visualization software.

The spatial partitioner demonstrated that the extraction and serialization of a product structure based JT file into an octree based spatial hierarchy can be effectively executed in a parallel fashion, however, it still left a lot to be desired in terms of overall performance and memory usage. While it was able to extract the 777 into an octree, it required that the process be run for approximately 12 hours overnight on a machine with over 24 GB of ram. The octree based spatial hierarchy provided a uniform spatial subdivision of triangles that worked well to demonstrate the benefits of GPU based occlusion culling. However, the uniform subdivision did not take into account the natural divisions within the model, and therefore, did not provide ideal spatial subdivision. The handling of large triangles fell short of expectations. Leaving them on higher-level cells resulted in a considerable amount of geometry being left too high in the spatial hierarchy and reduced its efficiency, while pushing them down into children cells resulted in far too much triangle duplication. Further, the actual splitting of occurrences into triangles runs contrary to established Product Data Management
(PDM) models and does not allow for on-the-fly configuration, or provide an easy means by which data access can be limited.

The spatial renderer showed that visibility guided rendering techniques have the potential to easily double previously established frame rates. It also showed that proposed algorithms have weaknesses. While the interleaving of cell rendering and occlusion query helps minimize stalls, it only works as long as there are cells available for rendering. The active cell heap became empty frequently, resulting in pipeline stalls while the renderer waited on occlusion results to become available. Interleaving ensures that visibility changes are reflected quickly (the algorithm will continue traversing until the complete visibility state is known,) however, this can lead to inconsistent frame rates, especially when the visibility of large sections of the spatial tree changes. Interleaving doesn’t readily separate render list render from render list generation, thus making it nearly impossible to reuse the results.
CHAPTER 3. SYSTEM ARCHITECTURE

The prototype proved that massive model technologies can significantly increase performance by rendering only triangles that are likely to contribute to the final image. However, its design left a lot to be desired, especially if it is to be integrated into an existing Computer Aided Drafting (CAD) or Engineering Visualization product. A formal project was executed as part of the Teamcenter Visualization 10.1 development cycle to adapt the prototype system developed as part of my master thesis to a system that will work directly against data stored in a Teamcenter Product Data Management (PDM) System. This work resulted in a new Spatial Hierarchy (SH) Design as well as a complete refactor in the Visual Determination Algorithm (VDA).

3.1 Spatial Hierarchy Design

The new spatial hierarchy is based on a bounding volume hierarchy over occurrences. Each cell within the tree contains its bounding volume information, occurrences, and children cells. The same occurrence can appear in multiple cells and occurrences contained within a cell can be dynamically determined at run time. No specific algorithm is currently required for partitioning of the spatial hierarchy, however several were implemented: Median Cut, Octree-Hilbert, and Outside-In.

The bounding volume over occurrence allows the visibility state of a given cell to be directly translated to the visibility state of an occurrence, and also allows the occurrences contained within a given cell to be dynamically configured as cells become visible. Both features are critical for integrating directly against a PDM and enabling visibility guided interaction. A spatial hierarchy over occurrences is severely limited in how it can spatially subdivide the model; the spatial hierarchy was updated to support the same occurrence in multiple cells to compensate for this, allowing for a better subdivision while still allowing for cell visibility to be traced back to individual occurrences. This also has the potential of enabling sub part culling in future versions of the architecture.
A query representation is dynamically generated for each cell at run time. This allows for the representation to be matched to the visibility determination algorithm used. For example, a set of triangles representing the bounding volume is generated with OpenGL occlusion queries.

### 3.2 Visibility Determination Algorithm

For any MMV solution to be viable it needs to limit the amount of data that is both loaded and rendered when viewing a scene. The visibility guided rendering algorithm first introduced in GPU Gems and further enhanced by Intervisuals3D performs well in determining the visibility of occurrences, and therefore limits the amount of data needed to be loaded. However, these algorithms run the risk of stalling the GPU if there are insufficient visible cells to avoid waiting for an occlusion query to complete. Worse, the interleaving of visibility determination with rendering makes it impossible to switch out the visibility determination algorithm or make full use of multi core or multi GPU systems. In order to get around these limitations, the visibility determination algorithm has been split into two pieces: Render List Generation and Render List Render.

---

**Figure 13: Visibility Determination Algorithm**
3.2.1 Render List Generation

Render list generation utilizes a visibility determination algorithm to create a list of all occurrences and their associated state that are likely to contribute to the current image. This is divided into five distinct stages under the current system architecture: Obtain Results, Update Culling, Update Lists, Render Depth, and Execute Query.

3.2.1.1 Obtain Results

During the obtain results stage the outstanding query list is traversed, and for all queries that are ready, the number of pixels that were hit during rendering are obtained from OpenGL. If the number of pixels is greater than the visibility threshold, the associated cell is marked as visible. The pixel value is also used to set the number of frames each cell should delay before executing another query. Frame delays for queries were introduced as a means to reduce the number of queries executed in a given frame. Originally these delays were set to a fixed value, however later research showed that a random delay is actually better at producing a more even spread [18]. The above logic aims for a middle ground in which cell queries are spread out: cells that are more likely to change visibility state are queried more frequently.

3.2.1.2 Update Culling

During the update culling stage tree cells are traversed, starting at the front and progressing up to the tree’s root cell. Any cell not contributing enough pixels to the current frame is marked as culled, and any cell whose children are all culled is marked potentially culled.

3.2.1.3 Update Lists

During the update lists stage the visibility value for all occurrences is initially set to 0. The tree is then traversed such that the pixels value for all visible cells can be propagated to their contained occurrences. The largest pixel value is used if an occurrence is referenced by multiple visible cells. Once traversal is complete, all occurrences with a pixel value greater than a preset threshold are harvested into a
render list with the level of detail selection for a particular occurrence based on its pixel value. The resulting render list is then sorted based upon material properties to minimize the amount of state changes that must occur whenever it is rendered.

### 3.2.1.4 Render Depth

During the render depth stage the render list from the previous frame is rendered slightly offset back into the depth buffer in order to initialize it for executing GPU based occlusion queries. The bound state during rendering is limited to only that state that can potentially influence the depth buffer results. Further transparent occurrences are ignored, as they are not likely to cause other occurrences to be completely occluded.

### 3.2.1.5 Execute Query

During the query execution stage the spatial hierarchy is traversed in a depth first order starting with the set of seed cells that are contained within the view frustum and moving all the way down to the visibility front. Each cell that is traversed loads its occurrence information if it is not already resident or is marked as invalid, determines if its children should be traversed, and if so inserts any that are not view frustum culled into the active cell heap, and finally executes the query action. Children cell loading and query actions are selected by combining relevant state to form an index value that can then be used to lookup the desired action from a carefully constructed table.

### 3.2.2 Render List Render

Render list render renders the list of all visible occurrence as efficiently as possible. The data structure was designed to utilize modern GPU functionality while minimizing the potential L2 cache impact. The current implementation is based around rendering unified vertex buffer objects (VBOs) (multiple shapes in the same buffer) with state information passed into the shader through uniform buffer objects (UBOs).
3.3 Baseline Performance

The performance of this system architecture was established by comparing the MMV rendering time to the time taken by the original LMV based approach using the same data. On average, the system is able to achieve approximately a 2x increase in performance, as seen in Figure 14, which is in line with the prototype; however, it is still far from achieving the desired results.

![Figure 14: LMV versus MMV performance under current system architecture](image)

Taking a closer look at the individual stages for two particular models, as shown in Figure 15, one can see that the CPU time is primarily dominated by the stages associated with the visibility determination algorithm. For the aircraft, over 72 percent of CPU time is spent in strategy. Of this, 24 percent of the time is spent populating the render list, 22 percent retrieving the results back from GL, 9.9 percent executing the OpenGL occlusion queries, and 7.2 percent traversing the spatial hierarchy to determine what cells need to be queried. For the cotton picker, over 80 percent of the CPU time was spent in strategy. Of this, over 31 percent is spent executing the OpenGL occlusion queries, 20 percent is spent populating the render list, and 14 percent is spent traversing the spatial hierarchy. The traversal, query, and result stages are all bound by the current visibility determination algorithm and could potentially be optimized by shifting to a more efficient approach. The population stage is responsible for generating...
a render list from the current visibility information and is likely to see performance gains only by decreasing the number of occurrences marked as visible.

![Percentage of CPU Time](image1)

**Figure 15: Percentage of CPU Time in each Stage for Generation 1 Algorithm**

The GPU Time, shown in Figure 16, shows a wide variation between models. For the aircraft dataset only 35 percent of the time was spent executing the VDA and of that, over 25 percent was spent populating the depth buffer by rendering the previous render list. For the cotton picker dataset, almost 75 percent of the time was spent executing the VDA, however the depth buffer population was a mere 5 percent. Executing OpenGL occlusion queries and generating the MDEI buffers dominated with 30 percent of the time each. This shows that as model size increases, the GPU time is bound stronger by what is rendered than the VDA used to determine what is to be rendered. Some performance gain can be appreciated by improving the depth population; however, a majority of the gains are likely to be achieved by only reducing the amount of geometry rendered by the render list.

![Percentage of GPU Time](image2)

**Figure 16: Percentage of GPU Time Each Stage for Generation 1 Algorithm**
CHAPTER 4. GPU DEPTH REPROJECTION

This Computer Engineering and Human Computer Interaction Research demonstrates a GPU based approach for reprojecting the depth buffer from one view into another in order to accelerate GPU occlusion queries by exploiting the natural frame to frame coherency in the depth buffer.

A key principle of most modern day MMV solutions is frame-to-frame coherency, or the concept that a visible set of occurrences is not likely to change considerably between adjacent render frames. This principle serves as the backbone upon which most culling techniques are based: it allows a set of occurrences to be minimized when each frame needs to be tested, especially when combined with an acceleration data structure such as a spatial hierarchy. The same principle can be applied to depth information required for executing occlusion queries, as the depth buffer is also not likely to change significantly between frames. While some approaches have begun to utilize this concept, none are designed to operate purely on the GPU.

Early MMV approaches found that the depth buffer can be used to generate higher fidelity alternative representations such as texture depth meshes, and it can be used as either stand-ins for particular occurrences [19, 20, 21] or faces of a particular view cell [22, 3, 23, 24, 25]. This allowed system performance to be increased by significantly reducing the number of polygons rendered for each frame at the cost of an expensive preprocessing step that might not be amenable to all applications.

Modern game engines utilize depth buffer approximations to accelerate their visibility determination algorithms. Frostbite developed by DICE uses the CPU to rasterize a course zbuffer on the CPU, based upon at most 10,000 vertices worth of low polygon count occluder meshes, which is then used to cull objects in the scene using a screen space bounding box test [13]. In a similar fashion, CryENGINE 3 by Crytek generates a depth buffer approximation and then uses hierarchical occlusion culling to cull objects from the scene by testing against both axes aligned bonding boxes and
object oriented bounding boxes on the CPU. The approximation is generated by using a software rasterizer to generate a depth buffer on a PC. On the console they were able to retrieve the depth buffer from 2 to 3 prior frames. The depth buffer was reprojected into the current view to improve the quality of the occlusion tests, and a dither operation was used to close some of the gaps [14].

Depth information is often populated by rendering visible occurrences from the previous frame into the depth buffer for GPU based occlusion approaches [12, 6]. The cost is amortized completely into the cost for rendering the visible occurrence into the color buffer for interleaved algorithms in which visibility is determined while generating the color buffer. This adds to the cost of rendering the full opaque render list for algorithms that have split visibility determination from color buffer generation intended to minimize GPU stalls. On large models this can equate to well over a million triangles: far greater than the number of pixels on the screen. For a large air craft data set this translates into over 8 percent of the CPU time, and 33 percent of the GPU time spent on just executing the render depth stage of the spatial strategy.

4.1 Alternative Approaches

A set of visible occurrences will not change significantly between two adjacent frames when based upon frame-to-frame coherence. This principle serves as the foundation by which most iterative spatial strategy algorithms achieve significant performance gains, as it allows the visibility state of one frame to be used as a basis for the next frame. The same principle can be applied to depth buffers produced by two consecutive frames: the previous depth buffer can be reprojected from its view point to the current view point and create an approximation of the current depth buffer. Crytek used this concept to improve the accuracy of their CPU side occlusion tests on consoles [14, 15]. Since the cost of reprojecting the depth buffer is tied to the number of pixels and not dataset size, it provides a more efficient means for establishing the initial depth buffer for occlusion culling than rendering the previous render list, especially if it is done without the need to transfer the data back from the GPU.
There are several options for reprojecting depth values from one viewpoint into another to create an approximation of the current depth buffer. Depth values could be read back to the host and generate a traditional texture depth mesh, which in turn could be rendered from the current viewpoint and populate the depth buffer [19, 20, 21]. While relatively simple, the cost of reading the depth buffer back is far too expensive to be practical. The performance hit to read the depth buffer back immediately would defeat the purpose and cause delay because the depth buffer from 2-3 prior frames has a greater potential to introduce artifacts. The depth buffer could be treated as a point cloud easily transformed into the new viewpoint as part of a vertex shader. While this has the potential to be extremely fast, it would cause holes in the calculated depth buffer as points farther away from the viewpoint travel a greater distance as the model is rotated. While conservative from a visibility perspective, this has the potential to cause a significant amount of hidden geometry to be loaded as it is briefly marked as visible. An alternative to the point cloud is to render quads that span between pixels in the depth buffer and produce a depth buffer erring on the side of caution when hidden objects become visible. In a worst case scenario, this will cause a single strategy iteration of lag in the time it takes certain hidden occurrences to become visible. Figure 17 shows differences in the depth buffer among nominal, point cloud, and quad based approaches.

Figure 17: Resulting color buffer for nominal, without quad, and with quad
4.2 GPU Depth Reprojection Design

Changes to both the Render List Render and Render Depth Stages were required to support reprojecting the depth buffer from a previous frame. A frame buffer object (FBO) with a depth texture render target was used to capture the state of the depth buffer after the rendering of all visible opaque geometry was completed by blitting the buffer from the main frame buffer. Testing showed this to have a negligible impact on system performance. During the render depth stage the quad mesh described above was rendered using a vertex shader to dynamically transform all the vertices from the previous viewpoint to the current viewpoint using the values from the depth texture as the initial vertices depth offset. If during the fragment shader a fragment is detected as having had an initial depth value at the depth buffer maximum, it is discarded. This ensures that depth values are only propagated for pixels caused by rendered geometry.

![Figure 18: GPU Depth Reprojection](image)
CHAPTER 5. GPU BATCH QUERY

This computer engineering and Human Computer Interaction Research demonstrates how GPU based Atomic Writes can be combined with a Spatial Hierarchy to significantly increase the parallelism on the GPU while allowing for scalability to arbitrarily large datasets.

The current visibility determination method utilizes OpenGL occlusion queries to determine the visibility state of cells within a spatial hierarchy. The basic algorithm is to traverse a spatial hierarchy in a screen depth first order and execute an individual occlusion query for each cell whose visibility state is in question. This approach results in the alternative representations of each cell along the visibility front being individually rendered, as well as multiple state transfers to read back the results from the queries. Modern GPUs run optimally when processing large batches of data in parallel. In terms of render this means pushing as many triangles as possible in a single draw call, thus countering the way traditional occlusion queries are executed. It is well known that GPU occlusion queries constitute an expensive operation \[13, 26\]. AMD recommends the number of queries in flight be limited to hundreds, and NVidia recommends the number of queries be limited to thousands: much lower than what may be necessary for arbitrarily sized models. Venders, game developers, and researchers are aware of these limitations and have been pursuing alternatives.

There has been a recent resurgence in CPU based approaches in the game industry as the GPU is often viewed as a scarce resource best left for more important tasks such as rendering. Prime examples of this are FrostBite 2 and CryEngine 3 games engines. Both of these approaches use a software rasterizer on sub thread to execute screen space culling of objects based upon their AABBs or OBBs in a fashion similar to the hierarchical occlusion culling technique \[13, 14, 15, 27\]. One problem with these approaches is that they assume the increases in number of available CPU cores will help them perform as well or better than a silicone piece that has been highly optimized for handling this very problem, and whose performance gains have been far outstripping
the CPU for several years. Another problem with these approaches is they do not take into account that GPUs and their APIs are fast approaching the point of executing the entire culling and render list generation process on the GPU.

At the opposite end of the spectrum, there has been significant research into moving the entire culling process onto the GPU. Christoph Kubisch and Markus Tavenrath demonstrated a new approach for executing GPU based Occlusion Culling at GTC 2013. This approach utilized GPU Texture Writes in the fragment shader in order to execute batch occlusion query over a list of bounding boxes associated with occurrences [16]. This system was shown to significantly improve performance by allowing for increased parallelism on the GPU; however it also has several shortcomings. First, it does not provide controls for limiting data load based upon occlusion results. Second, the algorithm is designed to query a list of bounding boxes associated with occurrences, and therefore will scale linearly as the number of occurrences increases, severely limiting the size of data sets that can be handled. Third, queries are limited to bounding boxes which could potentially result in a significant amount of occluded geometry being marked as visible.

5.1 GPU Batch Query Design

The use of GPU Texture Writes in the fragment shader provides a strong basis for implementing an advance visibility determination approach. This system improves previous methods by executing queries over cells of a spatial hierarchy, instead of a list of occurrences in order to more efficiently handle massive datasets.

5.1.1 Shader Buffer Write

As part of OpenGL 4.2, specification users gained the ability to write to texture memory in the fragment shader, and this was further enhanced in GL 4.4 to include all buffers. Several new approaches to existing graphics problems have shown how this functionality can be used to significantly improve both performance and quality when compared to prior solutions. This system utilizes these atomic shader writes to
determine the number of pixels likely to be affected when rendering the contents of a given cell.

Spatial hierarchy is implemented as a vector of cells in which each cell can be uniquely identified by index. The structure of the hierarchy is established through each cell containing a parent index, a child index, and the number of children. In order to facilitate several algorithmic optimizations, the cells are defined in a depth first order, guaranteeing that children cells have a larger index and are grouped such that all cells within a sub-tree have contiguous indices, as shown in Figure 19.

A buffer containing integer values is allocated in parallel to the cells of spatial hierarchy. If a fragment associated with the bounding volume of a cell should pass the depth test when executing occlusion tests, the value contained within the associated index is incremented, resulting in the buffer containing the pixel hit count for all tested cells. A secondary buffer is allocated and persistently mapped such that values can be read back from it through a direct pointer access. At the end of each occlusion pass the values from the primary buffer is copied into the secondary buffer. This subtle enhancement allows the primary buffer to remain in GPU memory only, significantly improving the performance of atomic write operations.
5.1.2 Multi-Draw Indirect

Modern GPUs are optimized for executing work in parallel. The best performance is therefore achieved when processing large batches. From a render perspective this means packing as much geometry as possible into each draw call. In order to better facilitate this, several new draw calls have been added to the OpenGL specification, with one of particular interest formalized as part of OpenGL 4.3: `glMultiDrawElementIndirect`.

```c
struct DrawElementsIndirectBuffer
{
    DrawElementsIndirectBuffer()
    : count(0),
      instanceCount(0),
      firstIndex(0),
      baseVertex(0),
      baseInstance(0) {}

    UInt32 count;       // Number of elements to be drawn
    UInt32 instanceCount; // Number of instance to be drawn
    UInt32 firstIndex;  // Offset into index array
    Int32 baseVertex;   // Offset to vertex records
    UInt32 baseInstance; // Under GL 4.2 specifies the base instance for
                          // fetching instanced vertex attributes
};

void glMultiDrawElementsIndirect( GLenum mode,
                                  GLenum type,
                                  const void *indirect,
                                  GLsizei drawcount,
                                  GLsizei stride );
```

Figure 20: `glMultiDrawElementIndirect` Specification

For the system to make optimum use of the shader atomic write based occlusion tests specified above, it needs to be able to batch the rendering of cells bounding volumes while still able to uniquely identify the cell responsible for any fragments that are considered visible in the fragment shader. GL multi draw elements indirectly allow for multiple indexed geometric primitives to be rendered in a single draw call. More importantly, it provides an efficient mechanism for specifying each primitive readily accessed in both the vertex and fragment shader.
Geometric information for rendering bounding volumes of all cells is stored sequentially by the system in a single vertex buffer object (VBO) pair. A Multi Draw Elements Indirect (MDEI) buffer running parallel to the number of cells is initialized so that each DrawElementsIndirectBuffer contains the FirstIndex and BaseVertex for the geometric information of the corresponding cell in the VBO, and the BaseInstance is set to the corresponding cell index. This setup allows for either all cells or a sub-set of cells to be rendered in a single draw if desired, and allows the fragment shader to readily identify an associated cell for a given fragment.

![MDEI Buffer Layout](image)

**Figure 21: MEDI Buffer Layout**

### 5.1.3 System Integration

Execute Query, Obtain Results, and Update Culling Stages were modified in order to integrate batch query into the Render List Generation.
5.1.3.1 Execute Query

The execute query stage was updated to make use of texture write based occlusion queries. The entire spatial hierarchy is queried, rather than querying all cells along the visibility front. No tree traversal is therefore required, and with use of multi draw elements indirect, this is accomplished using a single draw call. The buffer used for capturing the per cells pixel values is rotated between three buffers to ensure there is no data collision between consecutive runs of the Render List Generation logic.

5.1.3.2 Obtain Results

The obtain results stage was modified to traverse the spatial hierarchy and update the pixel values of all cells. Children cells are only pushed into the stack for traversal if the cell currently being processed is consider visible and fully loaded. The latter is extremely important as it helps prevent an arbitrary amount of cells from being loaded when the visibility state of a given region is unknown.

5.1.3.3 Update Cell Visibility

The update cells visibility stage was eliminated as the visibility results of all cells are calculated for each strategy pass, and therefore there is no reason to propagate the results up the spatial hierarchy.
CHAPTER 6. USER STUDY DISOCCLUSION ARTIFACTS

Visibility determination algorithms estimate the visible set of occurrences when rendering from a particular viewpoint. While these estimates are designed to be conservative in nature, disocclusion artifacts often occur when initially establishing the visibility set and transitioning between viewpoints. This popping behavior is often earmarked as significantly reducing perceived quality when rendering scenes, however few studies have been conducted to reliably document how much popping is acceptable to users, especially when they are concentrating on primary task execution.

6.1 Study Design

In order to answer the above research question, a user study was conducted in which participants were presented with an animation rendered using different visibility determination algorithms. Participants were asked to select highlighted occurrences while watching the animation to better simulate a CAD or visualization environment, and to prevent them from focusing purely on quality. A brief participant survey was used to garner perceived quality after completion of the animation for each rendered algorithm.

6.2 Study Setup

The study presented participants with a carefully designed animation aimed at displaying common causes of disocclusion artifacts in a consistent manner for a set of visibility determination algorithms. Key aspects of the animation included: quick rotations, zooming in on detailed regions of the scene, moving inside objects, and moving through objects. Experience and experimentation had shown these key aspects to be significant triggers of disocclusion artifacts. Animation was chosen over video playback because it allowed for the selection of existing mechanisms.

Participants were asked during the study to select highlighted parts as they appeared in the animation to simulate search behaviors and actions that would occur
while utilizing a CAD program. At the same time they were encouraged to focus on something other than quality, anticipating that it might trigger a mild state of situational blindness [28]. Ten parts were randomly highlighted from a set of 15, providing for some consistency between different approaches and preventing users from preempting subsequent selections that might have changed. The time lapse between when an occurrence was first visible and when it was selected, as well as any misses, was captured for analysis.

Perceived quality was measured by presenting participants with a brief survey after rendering the animation for each of the visual determination algorithms. The survey was aimed at garnering the participants’ viewpoints on various points of interest, as detailed in Table 1. Of primary concern was the overall quality of various algorithms and potential artifacts noticeable to users.

<table>
<thead>
<tr>
<th></th>
<th>Post Animation Survey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>On a scale of 1 to 10 how would you rate the overall quality of the animation?</td>
</tr>
<tr>
<td>2</td>
<td>On a scale of 1 to 10 how would you rate the overall performance of the animation?</td>
</tr>
<tr>
<td>3</td>
<td>On a scale of 1 to 10 how would you rate the usability based upon the animation?</td>
</tr>
<tr>
<td>4</td>
<td>On a scale of 1 to 10 how prevalent was the load order of occurrences?</td>
</tr>
<tr>
<td>5</td>
<td>On a scale of 1 to 10 how would you rate the smoothness of the animation?</td>
</tr>
<tr>
<td>6</td>
<td>On a scale of 1 to 10 how prevalent were missing occurrences?</td>
</tr>
<tr>
<td>7</td>
<td>On a scale of 1 to 10 how prevalent were the visibility changes of occurrences?</td>
</tr>
<tr>
<td>8</td>
<td>On a scale of 1 to 10 how would you rate the ease of finding and selecting the highlighted parts?</td>
</tr>
<tr>
<td>9</td>
<td>How many highlighted parts did you see?</td>
</tr>
</tbody>
</table>

Participants for this user study were drawn from colleagues at Siemens PLM Software. This helped ensure that participants’ demographics would closely match the end user population and avoid potential confidentiality problems that might arise with external participants. While information from individuals with limited CAD experience was desirable, the need for non-disclosure agreement to protect the patentability of new visibility determination algorithms and the severely limited set of data that can be used with external users made this untenable. At the beginning of the study participants were present with a brief demographic survey, as shown in Table 2.
### Table 2: Participant Demographics Survey

<table>
<thead>
<tr>
<th></th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What is your level of education?</td>
</tr>
<tr>
<td>2</td>
<td>How much experience do you have with computers?</td>
</tr>
<tr>
<td>3</td>
<td>How would you rate computer proficiency?</td>
</tr>
<tr>
<td>4</td>
<td>How often do you play video games?</td>
</tr>
<tr>
<td>5</td>
<td>How would you rate your video game skill level?</td>
</tr>
<tr>
<td>6</td>
<td>How much experience do you have with Computer Aided Drafting or Computer Visualization?</td>
</tr>
<tr>
<td>7</td>
<td>How would you rate your skill level with Computer Aided Drafting or Computer Visualization?</td>
</tr>
<tr>
<td>8</td>
<td>How much experience do you have with Computer Graphics?</td>
</tr>
<tr>
<td>9</td>
<td>How would you rate your skill level with Computer Graphics</td>
</tr>
</tbody>
</table>

The consistency of animation playback between participants was attained by temporarily providing the same system to each participant. A Dell Precision M4800 with an Intel i7-4900, 32 GB of Memory, and an NVidia K2100M (355.85) was selected; it is representative of a system that might be found on a CAD user’s desk, and being mobile, was easily provided for each of the participants. The laptops power settings were set to performance mode and vertical sync was disabled to ensure the highest framerate possible.

The data selected for use in this study were associated with a simple aircraft model with just over 44 million triangles. The dataset was selected as it had significant detail on the interior of the aircraft normally occluded by the skin, and it is reasonably representative of datasets that MMV technology is targeted at handling. The model was instanced 6 times to bring the scene total to 264 million triangles in order to increase the complexity of the scene and to increase the likelihood of disocclusion artifacts.

---

**Figure 23: Approximate User Study Scene**
Five visibility determination algorithms were selected to establish a baseline and test depth coherency and texture write. The first algorithm selected was the original generation 1 algorithm. The next algorithm selected was current Generation 1 algorithm that had been optimized as a result of the Generation 2 implementation. Generation 2 was selected in order to test the effects of batch query independent of depth reprojection. Lastly, Depth Reprojection was tested with both Generation 1 and 2 Algorithms. The order in which the algorithms were viewed during the study was randomized to prevent it from influencing results.

Table 3: Visibility Determination Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Visibility Determination Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strict 1: Original Generation 1 Algorithm</td>
</tr>
<tr>
<td>2</td>
<td>Generation 1: Current Generation 1 Algorithm</td>
</tr>
<tr>
<td>3</td>
<td>Generation 2: Generation 2 Algorithm</td>
</tr>
<tr>
<td>4</td>
<td>Reprojection 1: Current Generation 1 Algorithm with Depth Reprojection</td>
</tr>
<tr>
<td>5</td>
<td>Reprojection 2: Generation 2 Algorithm with Depth Reprojection</td>
</tr>
</tbody>
</table>

The same path was used for all algorithms, as shown in Figure 24. The path was designed to highlight both strengths and weaknesses of Batch Query and Depth Reprojection. A full rotation around the exterior of the plane demonstrates how slow transitions are handled, while moving between the inside and the outside of the aircraft helps show the quick transition that occurs during a single frame in which the entire visibility state changes.
CHAPTER 7. RESULTS

Using standard CAD workstations, novel improvements in GPU based occlusion query strategy were researched for accuracy, performance, and perceived quality. A system with dual Intel Xeon x5355 Processors, 8GB of Ram, and an NVidia Quadro K5200 (361.75) was used to evaluate accuracy and performance, and a Dell Precision M4800 with an Intel i7-4900, 32 GB of Memory, and an NVidia K2100M (355.85) was used to execute the user study. Novel improvements were tested independently and combined, resulting in four primary test conditions, as show in Table 4. A fifth condition was added for the user study to represent the state of the original occlusion query based algorithm present when this research project was originally proposed.

Table 4: Approaches Used for Testing

| Generation 1 | The original occlusion based algorithm for executing occlusion queries. |
| Reprojection 1 | The original occlusion based algorithm with depth reprojection used for generating the depth buffer. |
| Generation 2 | The new batch query based algorithm for executing occlusion queries. |
| Reprojection 2 | The new batch query based algorithm with depth reprojection used for generating the depth buffer. |
| Strict 1 | The original occlusion query based algorithm with an additional depth pass to render the culled cells into the depth buffer. |

7.1 Accuracy

Accuracy of any approach can be defined in terms of 3 measures: missing occurrences, extra occurrences, and loaded occurrences. Missing occurrences is the number of occurrences that should have been marked as visible and not detected as such. This is the most important measure, because not only do missing occurrences potentially prevent users from carrying out desired operations in the system, they are very likely to result in the system being perceived as broken. Extra occurrences is the number of occurrences that were marked as visible but did not actually contribute to any pixels on the screen, thus resulting in wasted work when rendered. Additionally, the resolving of these occurrences could mean a significant number of extra frames before the visibility determination algorithm signals that it has completed the task.
Loaded occurrences are the number of occurrences loaded while determining the visibility state of all cells in the spatial hierarchy. Whereas the previous two metrics measure the state once the final answer is achieved, this one aims at measuring the path taken to get the final answer. Loading occurrences that are not truly visible waste memory, and have the potential to significantly increase the time needed to reach the final answer.

Accuracy was first evaluated by manual inspection to detect missing occurrences that might be perceived by the user. Accuracy was then evaluated by allowing the spatial strategy to run from the initial load until it determined it was done, and then comparing the visibility determination algorithm results against actual visibility. Depth Reprojection was evaluated by an additional accuracy check in which the projected depth was rendered on top of the actual geometry to validate that it produced a conservative approximation of the depth buffer.

7.1.1 Manual Inspection

Several sample scenes were loaded and investigated by utilizing the same traversal patterns with various approaches in order to initially evaluate accuracy. The primary objective of this investigation was to identify instances in which visible occurrences were not properly rendered. Of particular interest were missing occurrences, temporary flashing occurrences, and oscillating occurrences.

The first generation GPU based occlusion query strategy uses an iterative approach for generating a render list of all visible occurrences in a scene. By design, it is conservative in nature; it is better to render culled geometry and risk loosing a little performance than to not render all visible occurrences. However, as an iterative algorithm disocclusion, artifacts are more pronounced in that it can take several frames to establish the visibility state of all cells in the spatial tree. Under the original strict algorithm this problem was extremely pronounced, especially in regions where there was significant overlap of cells within the spatial hierarchy such as the interior of an aircraft. Under the right conditions this could cause the visibility determination
algorithm to stop before it has fully resolved the visibility state, or even worse, cause occurrences to be incorrectly labeled as culled. While the current Generation 1 approach significantly alleviates this problem, it still incorrectly labels visible occurrences as culled when the viewpoint approaches some sheet bodies.

The second generation GPU based approach uses a batch query that enables it to determine accurate visibility state for all cells within at most 2 iterations of the visibility determination algorithm. The new algorithm does not exhibit the same artifacts when approaching sheet bodies; however, moving through them can be problematic as the depth state on the other side is largely unknown. The iterative nature of generation 1 allows it to side step the issue as the new depth buffer can be established before the visibility state is marked as having changed significantly. By design, generation 2 queries the visibility state of all cells in the spatial hierarchy, and thereby has a brief surge in the number of visible occurrences while the depth buffer is re-established. The shader based implementation used for calculating the per-cell pixel values for generation 2 tends to exhibit more instability when viewing long, thin parts from a distance, thus resulting in oscillation in the visibility state.
Depth reprojection appears to have limited impact on overall accuracy. There is no discernable difference in the number of disocclusion artifacts, even along edges of opposite rotations, but it does have a tendency to cause the visibility state of coplanar planes to be incorrectly determined. This problem is easily mitigated by increasing the amount of polygon offset used when reprojecting the previous frames depth buffer; however, this comes at the expense of increasing the number of cells deemed visible.

Figure 27: Depth Reprojection Plane Z-Fighting

7.1.2 Complete Initial Render

A common metric used by companies to measure the performance of a visualization system is the time it takes to render the first complete frame. With LMV based approaches users were often faced with a significant initial delay between when a model was opened and when the user could start to interact with the scene, and worse, an even more significant secondary delay waiting for all the geometry to be loaded. MMV approaches improve upon this by allowing interactivity from the start, and limiting the data load to only the geometry likely to contribute to the final image. This requires a careful balancing between being ultra-conservative and potentially loading too much, and ultra-strict and potentially loading too little.

The first complete frame for a given model was loaded and cycled through a redraw loop until the render list was marked as final by the spatial strategy in order to measure its timing and accuracy. Performance was measured by capturing the time and number of frames between the initial load and the time the final frame was complete. Accuracy was measured by comparing occurrences that were loaded, with
ones that were ultimately marked as visible against the set of occurrences known to be visible.

The performance results from this test are arguably invalid. The primary factor is that the time needed to complete the first complete frame is bound by the data load, not by the spatial strategy used to trigger the load. A secondary and equally important factor is that the data load mechanism is not designed to properly handle an extremely aggressive strategy. It will slow down significantly if load requests are made in quick succession. For the Cotton Picker data set, the time and number of frames were approximately the same. For the Air Craft data set, Generation 2 was 27% slower than Generation 1; however, it rendered over 3 times the number of frames.

In terms of cotton picker data set accuracy, the reprojection was shown to load significantly less data and produce fewer false positives for visibility. Unfortunately, it also missed almost twice as many occurrences that should have been marked as visible in the final render list. The generation 2 approach resulted in a significant increase in the number of extra occurrences, but it managed the load in less than half the number of occurrences. It also missed the fewest amount of occurrences. The addition of Depth Reprojection to Generation 2 did not fare well; the number of missing occurrences increased, although it did not report as many extra ones, and it loaded almost half as many occurrences relative to Generation 1.

<table>
<thead>
<tr>
<th>Table 5: Cotton Picker Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
</tr>
<tr>
<td>Generation 1</td>
</tr>
<tr>
<td>Reprojection 1</td>
</tr>
<tr>
<td>Generation 2</td>
</tr>
<tr>
<td>Reprojection 2</td>
</tr>
</tbody>
</table>

For the air craft data set, reprojection was also shown to load significantly less data and produce fewer false positives for visibility. While it missed more visible occurrences than using the previous render list to populate the depth buffer, it was nowhere as significant as the cotton picker. Generation 2 was both a hit and a miss. By
itself it faired far worse than Generation 1, however, when combined with reprojection it produced by far the best results.

<table>
<thead>
<tr>
<th>Table 6: Air Craft Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
</tr>
<tr>
<td>Generation 1</td>
</tr>
<tr>
<td>Reprojection 1</td>
</tr>
<tr>
<td>Generation 2</td>
</tr>
<tr>
<td>Reprojection 2</td>
</tr>
</tbody>
</table>

7.1.3 360 Degree Rotation

The manner in which approaches handle changes in the visibility state as they move between frames is as important as the complete initial frame. Figure 28 shows how the number of visible occurrences and rendered triangles change per frame for Generation 1, Generation 2, versus the actual number of visible occurrences and triangles as determined by the visibility simplifier. Fewer occurrences and triangles mean the system should achieve a high framerate. For Cotton Picker and Air Craft data sets, Generation 2 consistently identified fewer occurrences (30.0 and 27.4 percent reductions, respectively) as visible when compared to Generation 1, although it still fell short of the actual visible number. The same is true for triangles, however for the Cotton Picker, the triangle reduction was 53.3 percent, far greater than the reduction in occurrences, and for the Air Craft it was 34.4 percent, only slightly higher. The additional triangle reduction is likely the result of LODs.
An interesting artifact which shows up in graphs is that Generation 1 requires several frames before reaching a stable state. As shown in the missing occurrence graphs in Figure 29, Generation 1 incorrectly labeled a significant number of occurrences as culled during the initial couple of frames, a natural side effect of the approaches iterative nature. In the Cotton Picker dataset there appears to be more missing occurrences than for generation 1, which is validated by ANOVA two factor without replication ($F=164.244$, $p=0.00$). For the Air Craft there does not appear to be much difference, however ANOVA two factor shows significance even after accounting for the initial couple of frames ($F=17.752$, $p=0.000$). G1 is shown as having a significant number of extra occurrences when compared to G2 for both data sets.
Figure 29: Missing Occurrences (Left) and Extra Occurrences (Right) while Rotating 360

7.1.4 Depth Buffer

Figure 30 shows the results of using the quad based reprojection of the depth buffer by rendering the resulting depth mesh over the original mesh when rendered with increasing amounts of rotation. The reprojected depth is completely obscured by the original geometry when it is rendered from the original viewpoint. At 1 degree of rotation one can barely see the reprojected depth along the edge opposite the direction of rotation. As the amount of rotation increases, the reprojected depth becomes easily discernable, being visible along every leading edge at 4 degrees of rotation.
Performance was evaluated by measuring the overall, per-frame, and per-stage times on various models for each of the proscribed test conditions. Two scenarios were used to collect these measurements: the first was to render an animation that rotated the model 360 degrees on the z-axis, and the second was to render an animation that traversed the same path as the one used in the user study.

7.2.1 360 Degree Rotation

In order to obtain a quick and reliable performance measurement, the model was rotated 360 degrees around the y-axis in 1-degree increments, and per-frame and overall statistics were collected. The scene was rotated until all geometry was resident prior to collecting any data to ensure consistency when collecting data from different approaches. This helped limit the scope of comparisons to just differences in algorithms, and ensured that outside influences such as configuration and load were kept to a minimum.
The overall performance of the 360 degree rotation for the Generation 2 approach was consistently higher than that for Generation 1, although the difference in performance was highly dependent upon the original performance achieved with LMV, as shown in Figure 31. The data sets were sorted so that those achieving the highest framerate when rendered using LMV are on the left and the lowest are on the right. If LMV techniques were capable of achieving high performance (greater than 120fps) on the data set, they saw little to no gain when rendered using various MMV approaches. If LMV only achieved moderate to low performance, the MMV approaches were shown to behave 1.5 to 2 times better than what was achieved by LMV. There was one instance when performance decreased while using the Generation 2 approach, which interestingly occurred on the only data set to reference part files containing monolithic subassemblies. Reprojection did not fare as well as Generation 2 when it came to data sets upon which LMV techniques achieved high performance. For these data sets the reprojection algorithm was incapable of rendering the depth buffer faster than what was achieved by rendering the render list. In the worst-case scenario, performance fell to 75 percent of Generation 1. It is a different story for datasets that achieved moderate to low performance with LMV techniques. When used in conjunction with Generation 1, performance increased by 1.2 times on average, and when used in conjunction with Generation 2 it increased 1.2 times, and then steadily increased to 2.5 times.

![Figure 31: Speed Up When Rendering 360 Degrees Relative to Generation 1](image)

It can be seen by looking at the render time associated with each frame for two specific models, as shown in Figure 32, that the increase in performance is spread throughout the entire 360, and not limited to any particular frame or set of frames. By
design, the strategy runs for every other frame, making it possible to distinguish the benefits associated with the increased strategy performance verses the resulting render list. The strategy associated with the Cotton Picker resulted in significant gains, whereas the render list only saw marginal gains for a fraction of the frames when comparing Reprojection 1 to Reprojection 2. The strategy used for the Air Craft again saw significant gains in performance, whereas the render list seemed to fair worse for most of the frames.

![Figure 32: Per Frame Performance for the Cotton Picker (Left) and Air Craft (Right)](image)

The number of render items produced by the strategy employed with each frame is shown in Figure 33. In general, fewer render items means fewer triangles, and therefore higher render performance. The highest performing approach is therefore the one that generates the smallest render list for a given viewpoint in the shortest amount of time, while ultimately still producing a valid result. Reprojection 2 can be shown to consistently produce a render list that is 25 and 22 percent respectively smaller than Reprojection 1 for both the Cotton Picker and the Air Craft.
Figure 33: Per Frame Render Items for the Cotton Picker (Left) and Air Craft (Right)

Figure 34 shows the rate of change in the number of render items for Reprojection 1 and 2 for both the Cotton Picker and Air Craft. The rate of change theoretically shows how quickly an algorithm is able to adjust to changes in a viewpoint. An efficient algorithm would show brief spikes followed by relative periods of calm. In the case of the 360-degree rotation, the visibility state is in a constant state of flux, and therefore will not exhibit significant regions of calm, as shown in the graphs below. Overall, both algorithms follow a similar trend in how they to adapt to changes in the visibility state of items in the render list. On average, Reprojection 2 does show a higher rate of change in the number of render items. This is expected behavior, as Generation 2 is able to update the visibility state of the entire spatial hierarchy in one iteration, whereas Generation 1 must traverse through one level of spatial hierarchy at a time.

Figure 34: Rate of Render Item Change for the Cotton Picker (Left) and Air Craft (Right)

The overall performance number can be broken down by individual stages, as shown in Figure 35 and Figure 36. This allows for a detailed analysis of whether new approaches achieved desired changes in behavior. For both the Cotton Picker and the
Air Craft, Reprojection all but eliminated the time spent generating the depth buffer with the previous render list. The overall effect was more pronounced for the Air Craft, as it spent significantly more time rendering the previous render list. Generation 2 significantly impacted several stages. The Traversal and Query stages were all but eliminated for both data sets. For the Cotton Picker the time spent in the population stage was reduced by over a factor of 3, however the obtain results stage unfortunately took 3 times as long. For the Air Craft, both the population and obtain result stages saw slight performance improvement.

Figure 35: CPU Performance of Cotton Picker for 360 Degree Rotation

Figure 36: CPU of Air Craft for 360 Degree Rotation
7.2.2 User Study Path

While the 360-degree rotation provided a good high-level performance overview, it did not take into account behavior variances as the user navigated through the scene. Statistics were collected while running the user study animation for each of the approaches to secure a more accurate picture. The user study was designed to demonstrate scenarios in which algorithms were likely to fail, and it should provide a more realistic view when interacting in the real world.

Table 7: Time to Complete User Study Animation

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>S1</th>
<th>G1</th>
<th>R1</th>
<th>G2</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>209.97</td>
<td>216.20</td>
<td>181.32</td>
<td>187.91</td>
<td>158.11</td>
</tr>
</tbody>
</table>

Table 7 shows the total time taken to complete the animation for each of the approaches. During playback, a minimum time between key frames was enforced to ensure a smooth and consistent animation when approaches were capable of rapid rendering as fast, or faster than when frames were originally captured. Reprojection 2 took significantly less time than other approaches, implying that of all the approaches it was the most capable of rendering at the same frame rate as key frame capture. It was followed by Generation 2 and Reprojection 1, taking 14 and 18 percent more time, respectively. The slowest approaches were Strict 1 and Generation 1, taking 33 and 37 percent more time to complete the animation. These results are in line with expectations and help to corroborate the performance numbers from the 360-degree rotation.

The time taken to reach each of the key frames was plotted, as shown in Figure 37, to determine if the time difference was isolated to a certain segment of the animation or spread throughout the entire playback. The graph shows that there was a significant difference in the rate at which key frames were produced for approximately the first 900 frames, at which point the approaches appear to have taken the same amount of time to complete the remaining frames.
The process of separating frames based upon key events and then renormalizing within those regions provides a clearer picture of how algorithms behave relative to one another, as shown in Figure 38. The first graph that represents the model rotating 360 degrees shows the same delineation as the overall graph. R2 renders on average at 14.268 fps, followed by G2 at 10.530 fps, R1 at 9.558 fps, S1 at 7.960, and G1 at 7.217. The next three graphs, walking up to the 1st plane, panning to look at each of the planes on the sides, and the initial walk inside, show convergences into 3 primary groups, with R2 rendering at 12.5 fps, R1 and G2 at approximately 9.8fps, and G1 and S1 rendering at 7.8 fps. The remaining graphs show that once entering and exiting the planes is begun, the algorithms are in lock step in terms of how long it takes them to reach key frames— at least until the animation enters the final outside stage, at which point they appear to be falling back into the original groupings.

Another way of approaching performance metrics is to visualize the number of actual frames rendered between each of the key frames, as shown in Figure 39. A value of 1 implies the algorithm rendered at the same or slower rate than that specified by the key frame, whereas a value greater than 1 implies the algorithm was able to render
faster. As shown by the graphs for R1 and R2, algorithms are slower than key path frames for the first 900 frames, and then again for approximately the last 300 frames. Combining this information with the previous graphs, it can be inferred there is little delineation between algorithms when they are capable of rendering faster than specified key frames.

![Figure 39: Number of Frames per Path Frame During User Study](image)

Figure 40 shows the time necessary to render each frame for various algorithms demonstrated in the user study. The best algorithm is one that consistently produces the lowest frame time, as it will update its visibility state quicker and produce a smoother animation. As demonstrated in the graph below, none of the algorithms were consistently better than the others when rendering the animation for the user study. During the initial 360-degree rotation, Generation 2 and Reprojection 2 had decidedly the best algorithms. Once the animation began transitioning between the inside and the outside of the plane, the performance difference between algorithms became significantly less clear. For the most part, Reprojection 1 and 2 held the lead with the same performance, however there were times during which Reprojection 1 pulled ahead and Reprojection 2 all but failed. Unlike the 360-degree rotation, reprojection appears to have had the greatest influence on performance when rendering the user study animation. Generation 2 appears to only have a perceivable impact when viewing the scene from a distance, or when the visibility front was relatively large. This is countered by the fact that Generation 2 saw a significant performance drop when
moving into regions of space in which the depth buffer had not been previously established.

![Per Frame Redraw Time](image)

**Figure 40: Redraw Time of User Study Path**

One of the proposed key advantages of the Generation 2 algorithm is its ability to update the visibility of all cells in fewer frames than the previous algorithm. Figure 41 shows the number of Render Items marked as visible as the user study animation is executed with each of the algorithms. From this graph it is hard to draw any conclusions about the reaction time, as there is a significant amount of noise. The G2 algorithm is shown to consistently produce fewer or the same amount of render items as the previous algorithm. It is also shown to spike on several occasions, likely the result of walking through geometry that obscured the entire viewpoint. On a similar note, depth reprojection is shown to cause a slight increase in the number of render items. This is not unexpected as it does produce an approximate depth buffer, and therefore is likely to produce false positives.
From a performance perspective, Depth Reprojection was meant to eliminate the cost of rendering a depth buffer during the spatial strategy, and Generation 2 was meant to significantly improve the performance of several key spatial strategy stages. Figure 43 shows the per-stage cost for both the CPU and the GPU of Reproductions 1 and 2. The graphs clearly show that the time to render the initial depth buffer is almost nonexistent. A comparison of the graphs clearly shows that the time to execute occlusion queries is significantly less for Reproduction 2, as it is based on the Generation 2 algorithm. Both sets of graphs demonstrate that approaches are ultimately bound by the amount of time spent processing work on the GPU. The CPU graphs spend a substantial amount of time waiting on the GPU to finish processing so that the front and back buffers can be swapped. The GPU graphs show that a large part of this is a consequence of visible geometry rendering, and it applies more to R1 than R2 because the cost of querying is relative to the amount of visible items. The cost of R2 query is fixed and can thus become substantial when the number of visible items is small. This can be mitigated to some degree by enabling a partial query of cells in the G2 approach. Similarly, the time spent rendering visible geometry can be shortened by supporting sub-part culling.

Figure 41: Number of Render Items Produced by each Algorithm during User Study
Figure 42: User Study Path Performance Reprojection 1

Figure 43: User Study Path Performance for Reprojection 2
7.3 Perceived Quality

Perceived quality was evaluated by presenting participants with an animation for each of the proscribed strategies, and result in a single primary explanatory variable with five levels: S1, G1, R1, G2, and G2. Users were also presented with a brief demographic survey that provided an additional set of nine auxiliary explanatory variables with four to six levels each. At the completion of each animation they were asked to complete a survey, which resulted in nine responsive variables per strategy. An analysis of variance (ANOVA) was used to determine if responsive variables were significantly different and if formal conclusions could be drawn. A Pearson r Correlation was used to determine if relationships existed among variables.

7.3.1 Survey Demographics

Participants were limited to Siemens PLM Software employees to protect intellectual property rights. This severely limited the population from which individuals could be drawn, ultimately resulting in only 18 users participating in the study. The education level of the study population was well above average, with 39 percent having earned a PhD, 33 percent having earned a MS, 22 percent having earned a BS, and only 6 percent holding a High School Diploma. The population was also heavily skewed in terms of experience, with almost 90 percent having 11+ years of experience with computers and over 60 percent having g 11+ years of experience with visualizations and graphics. The reported level of knowledge of CAD, Computer Visualization, and Graph was more evenly distributed, but still resulted in over 50 percent of the participants with advanced levels or better. Most participants were casual gamers, with 33 percent classifying themselves as novice, and an additional 39 percent classifying themselves as intermediate.

The participant population is a good approximation of what one would expect at a well-established company that utilizes CAD and visualization software as part of their product development cycle. With such a company one would expect there to be an
established workforce with significant experience in their respective areas, however their level of knowledge would vary. If any demographic could be considered skewed it would be education level, as most companies are not likely to have as high a density of PhDs in their workforce. The lack of a significant number of participants having experience with hardcore games is unfortunate, as they may have a different perspective in rating overall quality.

![User Study Demographic Information](image)

**Figure 44: User Study Demographic Information**

7.3.2 Survey Response

Question 1 measured participant opinions of overall quality for each of the approaches. Figure 45 shows the survey results for all participants, separated by each approach. Based on means, Reproduction 1 (M=7.111, SD=1.183) had the highest overall quality, and Generation 2 (M=5.944, SD=1.798) had the lowest. Interestingly, Reproduction 2 (M=6.778, SD=1.592) had the second highest overall rating, implying that
the addition of depth reprojection was more than enough to offset the negative feelings users had with the atomic write based approach. ANOVA two factor without replication showed a significant difference ($F= 5.278, p = 0.018$) within the results, however post hoc Tukey HSD demonstrated the significant difference only existed between R1 and S1 ($p=0.038$).

![Figure 45: Question 1 Results (Left) Mean and Confidence Interval (Right)](image)

Question 2 measured each participant’s opinion of render speed for each approach overall performance. Figure 46 shows participant responses for each approach. Based on means, Reprojection 1 ($M=7.278, SD=1.183$) and Reprojection 2 ($M=7.278, SD=1.592$) tied for the highest overall performance, however there was significant more variance in the Reprojection 2 results. ANOVA two factor without replication again showed a significant difference ($F=7.797, p=0.001$) in results. Post hoc Tukey HSD showed significant differences among R1 and both G1 ($p=0.035$) and S1 ($p=0.008$), as well as R2 and both G1 ($p=0.035$) and S1 ($p=0.008$).

![Figure 46: Question 2 Results (Left) Mean and Confidence Interval (Right)](image)

Question 3 measured participants’ opinion on usability for each approach: “Did the disocclusion artifact prevent or deter you from completing the task?” Figure 47 shows approach results for all participants. Based upon means, Reprojection 2 ($M=6.994, SD=1.162$) achieved the highest result, followed closely by Reprojection 1
ANOVA two factor without replication showed insignificant differences (F=1.816, p=0.136), implying that participants did not perceive usability differences among any of the approaches.

Figure 47: Question 3 Results (Left) Mean and Confidence Interval (Right)

Question 4 measured participants’ opinion on occurrence load order for each of the approaches. Figure 48 shows the results for each approach. Based upon means, Generation 2 (M=6.222, SD=2.439) achieved the highest result. Its ability to quickly resolve visibility changes possibly acted as a double-edged sword. The iterative nature of Generation 1 caused it to slowly transition between visibility changes that may not have been as perceptible to participants as instantaneous changes that occur with Generation 2. This effect was exacerbated by the higher prevalence of level of detail switching (LOD) that also occurred with Generation 2. For Generation 1, only cells along the query front actively querying can change their current LOD during use. This means that a vast majority of visible occurrences will remain constant, and in the case of interior cells they will almost always use the high LOD. Since Generation 2 queries all visible cells in every frame, it is more prone to fluctuations in the LODs currently used. ANOVA two factor without replication failed to show a significant difference (F=1.412, p=0.239) in any of the results, thus while it is possible to explain why G2 could be perceived as having a worse load order, users did not differentiate it enough from other approaches to draw a firm conclusion that it was viewed accordingly.
Question 5 measured participants’ opinion on smoothness and consistency of framerate for each approach. Figure 49 shows participant results associated with the respective approaches. Based upon means, Reprojection 2 (M=7.000, SD=1.455) achieved the highest result, followed by Reprojection 1 (M=6.778, SD=1.215). ANOVA two factor without replication again showed a significant difference (F=7.797, p=0.000) in results. Post hoc Tukey HSD showed a significant difference between R1 and both S1 (p=0.001) and G1 (p=0.031), as well as R2 and S1 (p=0.000), G1 (p=0.007) and G2 (p=0.043).

Question 6 measured participants’ opinions about the prevalence of missing occurrences for each approach. Figure 50 shows respective participant results. Based on the means, Generation 2 (M=5.222, SD=2.713) achieved the highest result, implying that participants noticed missing occurrences most frequently with this approach. Strict 1 (M=4.000, SD=1.910) with the second lowest result had the highest number of cases in which visible occurrences were incorrectly kept as culled. ANOVA two factor without replication showed an insignificant difference (F=2.158, p=0.083) between approaches.
Question 7 measured participants’ opinions of the prevalence of visibility changes for each approach. Figure 51 shows the results of participant responses by approach. Based on means, Generation 2 (M=6.556, SD=2.202) achieved the highest result, followed by Reprojection 2 (M=5.994, SD=2.313). ANOVA two factor without replication failed to show a significant difference (F=2.338, p=0.064) between approaches.

Question 8 measured participants’ opinion on the ease of finding and selecting highlighted parts of each approach. Figure 52 shows the results garnered from participants by approach. Based upon the means, Reprojection 1 (M=6.222, SD=1.592), Reprojection 2 (M=6.111, SD=1.997), and Generation 2 (M=5.994, SD=1.984) were reported as having the best results, respectively. ANOVA two factor without replication failed to show a significant difference (F=1.042, p=0.392) in any of the results, implying participants did not clearly see a difference in usability among the approaches.
Figure 52: Question 8 Results (Left) Mean and Confidence Interval (Right)

Question 9 measured participants’ opinion on overall usability for each approach: “Did the disocclusion artifact prevent or deter you from completing the task?” Figure 53 shows the results from all participant responses associated with each approach. Based on means, Reprojection 2 (M=9.111, SD=2.374) achieved the highest result, implying that users were more likely to count the correct number of occurrence with R2 than with any other approach. ANOVA two factor without replication failed to show a significant difference (F=1.030, p=0.398) among approaches.

Figure 53: Question 9 Results (Left) Mean and Confidence Interval (Right)

7.3.3 Survey Rank

An analysis of the survey questions using the original one-to-ten rank scale showed that overall quality was not significantly different among most of the approaches. An alternative way of looking at the data was to translate survey responses into a rank system in which approaches were ranked from best to worse for each participant. This was accomplished by first sorting the algorithm according to overall quality, and then according to performance. Approaches were given the lowest rank in the case of a tie. Figure 54 shows the results of this process.
Figure 54: User Study Survey Results Based Upon Individual Ranking

Based on this translation, Strict 1 was never perceived as having the highest quality. Generation 1 was skewed slightly to the right, with most participants viewing it as having a lower quality. Reprojection 1 was heavily skewed to the left, with just under half of the participants viewing it as having the highest quality. Generation 2 was skewed heavily to the right, with well over half of the participants viewing it as having the lowest quality. Reprojection 2 was the most polarizing, with participants split into two groups: those who viewed it favorably and those who viewed it unfavorably. An ANOVA two factor without replication run against the rank data (F=5.069, p=0.001) showed a significant difference among approaches. A post hoc Tukey HSD on rank data showed R1 significantly better than S1 (p=0.014), G1 (p=0.027) and G2 (p=0.004).

7.3.4 Survey Correlation

For each of the approaches, a Pearson correlation test was used to identify linear correlations in survey results for the user study. For all approaches there was a positive linear correlation for overall quality, performance, smoothness, and usability. There was also a positive linear correlation between prevalence of load order and visibility changes of occurrences. For all approaches except Strict 1, there was a positive linear correlation between performance and ease of finding and selecting occurrences. Strict 1 showed a positive linear correlation between prevalence of missing occurrence and
visibility changes, while strict 1 and Generation 2 showed a negative correlation between usability and the prevalence of load changes. The prevalence of visibility changes and missing occurrences yielded a negative correlation for both overall quality and performance with Generation 2. No correlation was detected between participant demographics and survey results.

Table 8: Correlations within Survey Results

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strik 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0.816</td>
<td>0.572</td>
<td>0.770</td>
<td>0.574</td>
<td>0.821</td>
</tr>
<tr>
<td>p</td>
<td>0.000</td>
<td>0.013</td>
<td>0.000</td>
<td>0.014</td>
<td>0.000</td>
</tr>
<tr>
<td>Generation 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0.907</td>
<td>0.879</td>
<td>0.881</td>
<td>0.963</td>
<td>-0.447</td>
</tr>
<tr>
<td>p</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.003</td>
<td>0.001</td>
</tr>
<tr>
<td>Reprojection 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0.750</td>
<td>0.826</td>
<td>0.878</td>
<td>0.517</td>
<td>0.798</td>
</tr>
<tr>
<td>p</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Generation 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0.774</td>
<td>0.811</td>
<td>0.649</td>
<td>-0.625</td>
<td>-0.631</td>
</tr>
<tr>
<td>p</td>
<td>0.000</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Reprojection 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0.741</td>
<td>0.766</td>
<td>0.711</td>
<td>-0.526</td>
<td>0.808</td>
</tr>
<tr>
<td>p</td>
<td>0.000</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

7.3.5 Discussion

The animation failed to clearly delineate different algorithms from one another. The small population size limited the fidelity of the collected data, making it difficult to draw firm conclusions. The path used for the animation did not allow for different approaches to be easily delineated from one another. For the majority of key frames, algorithms achieved multiple frame draws, obscuring potential disocclusion artifacts. The data set chosen for the study lacked the needed complexity of target data sets. Although inserting additional instances into the environment helped to increase the complexity of the scene, it spread through a much larger area and significantly changed visibility front mechanics. Low LODs contained within the data set were not representative of the high LOD geometry, potentially causing severe artifacts when rendered. In essence, all of the above created an environment that was stacked against the generation 2 approach with its faster update speed and more reliable LOD selection. Depth reprojection fared well because of its significant increase in framerate.
CHAPTER 8. CONCLUSION AND FUTURE WORK

More effective culling techniques are needed if computer graphics are going to keep up with the current data explosion associated with ever increasing complex designs. Systems based upon occlusion culling approaches have been shown to be effective tools for extremely large data sets; however there is room for improvement. The present dissertation demonstrates that novel improvements in existing systems are effective in not only improving performance, but also reducing undesirable artifacts that occur when using culling techniques.

This research suggests that reprojecting the depth buffer from one frame into another to exploit frame-to-frame coherency is an effective alternative to the traditional approach of rendering the previous frames render list to populate the depth buffer. Whereas similar techniques require that the depth buffer is read back to the CPU for processing, this approach was able to make use of advance shaders to reproject the depth buffer from one view to another without the data ever leaving the GPU. On several data sets it was shown that this new algorithm completely eliminates the measured cost of producing the required depth buffer for occlusion culling. Future research could potentially improve upon the performance gains by (a) utilizing blit pixels when the view is detected as the same, (2) down sample the captured depth buffer into a smaller viewport at reprojection time, or (3) dynamically tessellating the rendered mesh to minimize the number of polygons required in regions of low depth complexity.

This research also demonstrates that atomic shader writes provide a viable alternative to traditional GL occlusion queries. Whereas traditional occlusion queries are limited only to querying a single entity at a time, atomic write can be used to query several entities in one go. The approach shows that atomic shader writes can be effectively used to query the visibility state of an entire spatial hierarchy in the time it would normally take to query only a handful of its cells. Furthermore, combining this with a simple blocking trick was shown to preserve the desired loading behavior present in the original algorithm. This area of exploration is far from complete. Future
research could finish splitting the spatial hierarchy into smaller query sets, enabling higher level sets to determine whether lower level sets need to be queried, or whether the set of cells being queried can be adjusted, based upon the current visibility front in effort to reduce the overhead on larger spatial hierarchies. Future research could also limit cell visibility state changes to only those cells within a certain distance from the front to further minimize disocclusion artifacts and potentially improve data load behavior.

The user study showed that perceived quality is influenced by far more than just disocclusion artifacts. While participants viewed the resulting framerate increase of depth reprojection extremely favorably, the same was not true for atomic write based occlusion queries. By itself, users overwhelmingly ranked Generation 2 as the worst though the simple act of combining it with depth reprojection was enough for it to be ranked as the second best approach. While the user study proved informative, it was far from perfect. Determining the quality of the animations was extremely subjective, with participants forced to rank them relative to one another without the ability to see them side by side or to go back. A better alternative would be to use choice modeling with animations displayed side by side. At the very least, a golden approach should have been presented with the render list precooked so as to establish a baseline by which all other animations could be compared.
BIBLIOGRAPHY


