Efficient Stereoscopic Rendering of Building Information Models (BIM)

Mikael Johansson
Chalmers University of Technology, Sweden

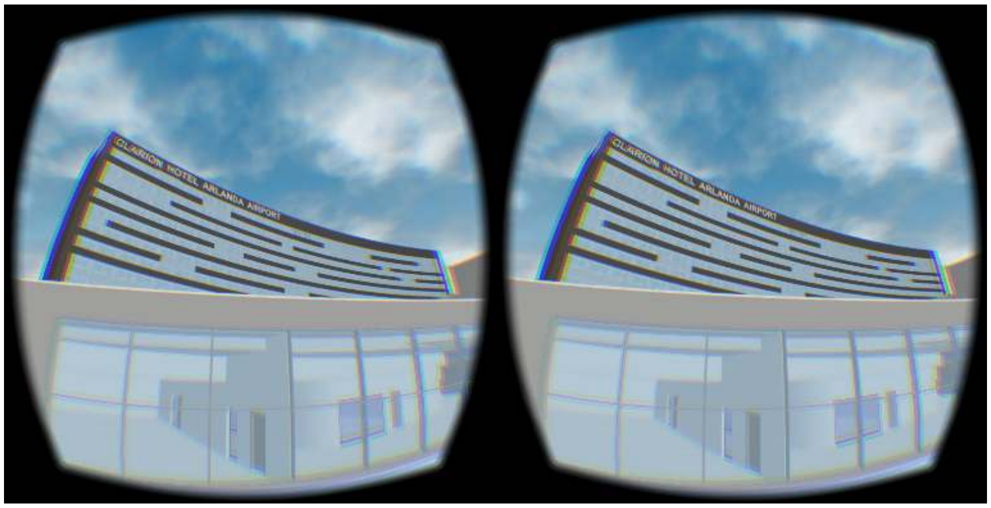

Figure 1. A building information model (BIM) taken from a real-world project rendered at more than 90 frames per second using the stereo rendering techniques presented in this paper.

Abstract

This paper describes and investigates *stereo instancing*—a single-pass stereo rendering technique based on hardware-accelerated geometry instancing—for the purpose of rendering building information models (BIM) on modern head-mounted displays (HMD), such as the Oculus Rift. It is shown that the stereo instancing technique is very well suited for integration with query-based occlusion culling as well as conventional geometry instancing, and it outperforms the traditional two-pass stereo rendering approach, geometry shader-based stereo duplication, as well as brute-force stereo rendering of typical BIMs on recent graphics hardware.

1. Introduction

The advent of building information models (BIM) and consumer-directed HMDs, such as the Oculus Rift and HTC Vive, has opened up new possibilities for the use of virtual reality (VR) as a natural tool during architectural design. The use of BIM allows a 3D scene to be directly extracted from the architect's own design environment and, with the availability of a new generation of VR-systems, architects and other stakeholders can explore and evaluate future buildings in 1:1 scale through an affordable and portable interface.

However, given the rendering-performance demands posed by modern HMDs, the use of BIMs in a VR setting remains a difficult task. With BIMs being primarily created to describe a complete building in detail, many 3D datasets extracted from them provide a challenge to manage in real-time if no additional acceleration strategies are utilized [Johansson et al. 2015]. In this context, a combination of occlusion culling and hardware-accelerated geometry instancing has been shown to provide a suitable option in a non-stereo environment [Johansson 2013]. This approach could be adapted to a stereo setup simply by performing two rendering passes of the scene, one for the left eye and one for the right eye. However, this would still require an additional and equal amount of rendering time, as the number of occlusion tests, rasterized triangles, and issued draw calls would increase by a factor of two.

In order to remove the requirement of a second pass, the concept of *stereo instancing* has recently gained a lot of attention within the game development community [Wilson 2015; Vlachos 2015]. Stereo instancing refers to a technique where hardware-accelerated geometry instancing is used to render left and right stereo pairs during a single pass.

In this paper, a more detailed description of the stereo instancing technique is provided together with a thorough performance evaluation using real-world datasets. Furthermore, the stereo instancing technique is used to extend the work in [Johansson 2013] in order to provide efficient stereoscopic rendering of BIMs. It is shown that stereo instancing works very well in combination with occlusion culling based on hardware-accelerated occlusion queries in a split-screen stereo setup. In addition, the use of hardware-accelerated instancing is generalized to support both replicated geometry as well as single-pass generation of stereo pairs.

2. Stereoscopic Rendering, Bottlenecks, and Acceleration Techniques

Any true stereoscopic display solution requires that the user be presented with different images for the left and right eye. In the case of the Oculus Rift, this is implemented with split-screen rendering, where an application renders the image for the left eye into the left half of the screen, and vice versa. However, due to the lenses, which provide an increased field of view, a pincushion distortion of the image is introduced. To cancel out this effect, the rendering has to be done at a higher resolution, followed by a post-processing step that performs a barrel distortion. In Figure 2, the Oculus stereo rendering pipeline is illustrated.

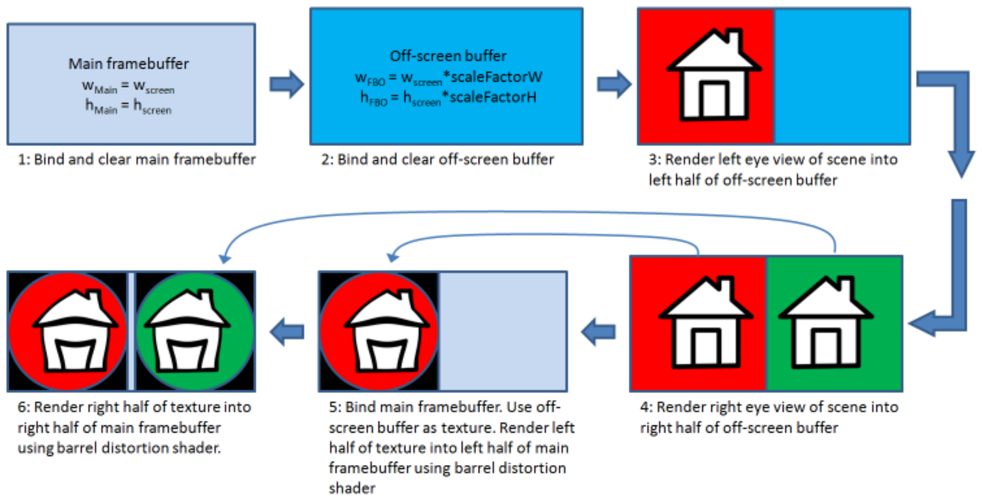
Main framebuffer
$w_{Main} = w_{screen}$
$h_{Main} = h_{screen}$
1: Bind and clear main framebuffer
Off-screen buffer
$w_{FBO} = w_{screen}$*scaleFactorW
$h_{FBO} = h_{screen}$*scaleFactorH
2: Bind and clear off-screen buffer
3: Render left eye view of scene into left half of off-screen buffer
4: Render right eye view of scene into right half of off-screen buffer
5: Bind main framebuffer. Use off-screen buffer as texture. Render left half of texture into left half of main framebuffer using barrel distortion shader
6: Render right half of texture into right half of main framebuffer using barrel distortion shader.

Figure 2. A typical stereo rendering pipeline for the Oculus Rift.

In addition to requiring two distinct views of the scene every frame, stereoscopic rendering targeting head-mounted displays (HMDs) typically has much higher interactivity demands compared to monoscopic, desktop applications. With the introduction of the consumer versions of both the Oculus Rift and HTC Vive, the minimum frame rate has been set to 90 Hz, which corresponds to a maximum frame time of 11.1 ms.

Traditionally, stereo rendering has almost exclusively been implemented as two independent, serial rendering passes, effectively doubling the geometry workload on both the CPU and the GPU. When considering ways to improve rendering performance in such a setup, we can generalize the situation in a similar fashion as Hillaire [2012]; if an application is CPU-bound, its performance will be mostly influenced by the number of draw calls and related state changes per frame. If, on the other hand, the application is GPU-bound, its performance will be influenced by the number of triangles drawn every frame and the complexity of the different shader stages.

As with monoscopic rendering, there are several different acceleration techniques that can potentially be utilized if the performance requirements are not met. Common examples include level-of-detail (LOD) to reduce the geometric complexity of distant objects; frustum, occlusion, or contribution culling to reduce the number of objects to render; hardware-accelerated instancing to reduce the number of draw calls when rendering replicated geometry; and geometry batching to render groups of primitives with as few draw calls as possible. An option more unique to stereoscopic rendering is to use the geometry shader to render left and right stereo pairs from a single geometry pass [Marbach 2009]. The geometry shader supports multiple outputs from a single input as well as functionality to redirect output to a specific viewport. Consequently, a single draw call per batch of geometry is sufficient in order to produce both the left and right eye projection. However, even if this reduces the CPU-burden—by reducing the number of draw calls as well as related state changes—the geometry shader typically introduces significant overhead on the GPU.

3. Stereo Instancing

As the name implies, stereo instancing takes advantage of hardware-accelerated geometry instancing in order to produce both the left- and right-eye version of the scene during a single rendering pass. With the instancing capabilities of modern GPUs, it is possible to produce multiple output primitives from a single input, without introducing the geometry shader. To support stereo instancing, an application only needs to replace conventional draw calls (i.e., `glDrawArrays`) with instanced ones (i.e., `glDrawArraysInstanced`), providing two (2) as the instance count. Based on the value of the instance counter in the vertex shader (i.e., `gl_InstanceID` being zero, repectively one), each vertex can then be transformed and projected according to the left or right eye, respectively.

However, the main difficulty with this approach is that unextended OpenGL currently does not support multiple viewport outputs from (within) the vertex shader. Fortunately, this can be solved by performing a *screen-space transformation* of the geometry, together with *user-defined clipping planes*. The screen-space transformation procedure used in this paper is similar to that in [Trapp and Döllner 2010], but it is performed in the vertex shader instead of in the geometry shader. As illustrated in Figure 3, the extent of a single viewport in OpenGL will range from $[-1, 1]$ in normalized device coordinates (NDC) for both the x- and y-coordinates. In order to

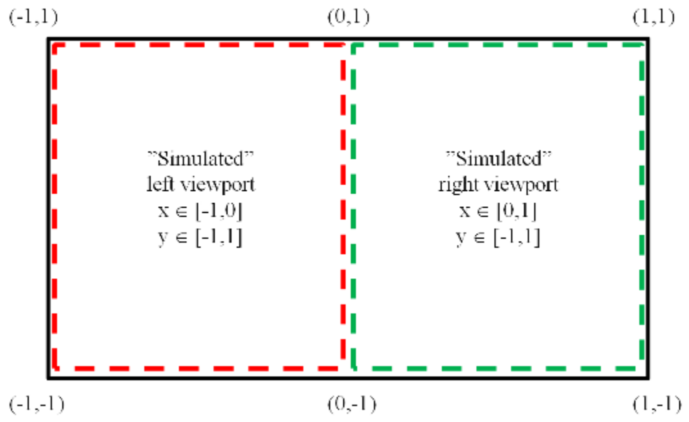

Figure 3. The extent of the simulated left (red) and right (green) viewports, as well as the real viewport (black) in normalized device coordinates (NDC).

simulate the concept of two side-by-side viewports in the vertex shader, the idea is to perform a screen-space transformation so that the x-coordinates of the left- and right-eye geometry will range from $[-1, 0]$ and $[0, -1]$, respectively, in NDC. Listing 1 provides GLSL vertex shader code for this process. A vertex *v (x,y,z,w)* is first transformed into 4D homogenous clip space (CS) by the left or right model-view-projection (MVP) matrix. This is followed by a division by the homogenous vector component *w* in order to further transform it into NDC. In NDC, the x-coordinate is then mapped from $[-1, 1]$ to either $[-1, 0]$ (left) or $[0, 1]$ (right). Finally, the NDC-transformed vertex has to be transformed back to CS. In addition, a user-defined clip plane is used to restrict rendering within either one of the two simulated viewports. The code in Listing 1 assumes that a single full screen viewport has been previously defined on the client side.

```
#version 430 core
layout (location = 0) in vec3 position3;
//ViewProj matrix (left and right):
uniform mat4 viewProjMatrixLeft, viewProjMatrixRight;
uniform mat4 modelMatrix; //The Model matrix
//Frustum plane coefficients in World-Space:
uniform vec4 leftEyeRightPlaneWS, rightEyeLeftPlaneWS;
void main() {
   vec4 vertPos = vec4(position3, 1.0);
   vec4 vertPosWS = modelMatrix * vertPos;

   if(gl_InstanceID < 1) { //Left eye
      vec4 v = viewProjMatrixLeft * vertPosWS; //Transform to CS
      vec4 vPosNDC = v/v.w; //...and further to NDC
      float xNew = (vPosNDC.x-1.0)/2.0; //X from [-1,1] to [-1,0]
      vPosNDC.x = xNew;
      gl_Position = vPosNDC*v.w; //Transform back to CS
      //Additional clip plane to the right
      gl_ClipDistance[0] = dot(vertPosWS, leftEyeRightPlaneWS);
   }
   else { //Similar code as above, but for the right eye
      vec4 v = viewProjMatrixRight * vertPosWS;
      vec4 vPosNDC = v/v.w;
      float xNew = (vPosNDC.x+1.0)/2.0; //X from [-1,1] to [0,1]
      vPosNDC.x = xNew;
      gl_Position = vPosNDC*v.w;
      gl_ClipDistance[0] = dot(vertPosWS, rightEyeLeftPlaneWS);
   }
}
```

Listing 1. GLSL vertex shader illustrating stereo instancing.

Depending on hardware, the screen-space transformation and custom clipping in Listing 1 can be omitted in order to simplify the vertex shader. Both Nvidia and AMD provide OpenGL extensions that allow access to multiple defined viewports in the vertex shader in a similar fashion as for the geometry shader (i.e., `NV_viewport_array2` and `AMD_vertex_shader_viewport_index`).

Still, as in the case with stereo duplication in the geometry shader, stereo instancing does not reduce the number of triangles that needs to be rendered every frame. As a consequence, the sheer amount of geometry that needs to be transformed, rasterized, and shaded every frame, may very well become the limiting factor.

3.1. Integration with Occlusion Culling and Conventional Instancing

In [Johansson 2013], a combination of occlusion culling and hardware-accelerated geometry instancing was used in order to provide efficient real-time rendering of BIMs. In essence, CHC++ [Mattausch et al. 2008], an efficient occlusion culling algorithm based on hardware-accelerated occlusion queries, was extended to support instanced rendering of unoccluded replicated geometry.

The original CHC++ algorithm takes advantage of spatial and temporal coherence in order to reduce the latency typically introduced by using occlusion queries. The state of visibility from the previous frame is used to initiate queries in the current frame (temporal coherence) and, by organizing the scene in a hierarchical structure (i.e., bounding volume hierarchy), it is possible to test entire branches of the scene with a single query (spatial coherence). While traversing a scene in a front-to-back order, queries are only issued for previously invisible interior nodes (i.e., groups of objects) and for previously visible leaf nodes (i.e., singular objects) of the hierarchy. The state of visibility for previously visible leaves is only updated for the next frame, and they are therefore rendered immediately (without waiting for the query results to return). However, the state of visibility for previously invisible interior nodes is important for the current frame, and they are not further traversed until the query results return. By interleaving the rendering of (previously) visible objects with the issuing of queries, the algorithm reduces idle time due to waiting for queries to return.

To integrate hardware-accelerated instancing within this system, the visibility knowledge from the previous frame is used to select candidates for instanced rendering in the current frame. That is, replicated geometry found visible in frame n is scheduled for rendering using instancing in frame $n + 1$. For viewpoints with many objects visible, the proposed solution was shown to offer speed-ups in the range of 1.25x-1.7x compared to only using occlusion culling, mainly as a result of reducing the number of individual draw calls.

When considering ways to adapt the work presented in [Johansson 2013] to a stereo setup, the properties of the stereo instancing technique offer great opportunities. As it turns out, the combination of single-pass stereo-pair generation, hardware-accelerated occlusion queries, and a split-screen setup becomes an almost perfect fit. With a single depth buffer used for both the left and right eye, only a single occlusion query is ever needed per visibility test. That is, when a bounding box representing an interior or leaf node of the spatial hierarchy is simultaneously rendered to both the left and right viewport, a single occlusion query will report back a positive number if either one of the representations (i.e., left or right) turns out to be visible. So, compared to a two-pass approach, not only the number of draw calls and state changes but

```
#version 430 core
#extension GL_EXT_gpu_shader4 : require
#extension GL_NV_viewport_array2 : require
layout (location = 0) in vec3 position3;

//ViewProj matrix (left and right):
uniform mat4 viewProjMatrixLeft, viewProjMatrixRight;
uniform samplerBuffer M; //Matrices
uniform samplerBuffer I; //Indices
uniform int offset; //Per-geometry type offset into I

void main() {
   int leftOrRight = gl_InstanceID%2;
   float glInstanceIDF = float(gl_InstanceID);
   float instanceID_Stereo = int(floor(glInstanceIDF/2.0));
   int instance_id_offset = offset+instanceID_Stereo;
   float offsetF = texelFetchBuffer( I, instance_id_offset).x;
   float startPosF = offsetF*4.0;
   int startPosInt = int(startPosF);

   mat4 modelMatrix = mat4(texelFetchBuffer(M,startPosInt),
                           texelFetchBuffer(M,startPosInt+1),
                           texelFetchBuffer(M,startPosInt+2),
                           texelFetchBuffer(M,startPosInt+3));

   vec4 vertPos = vec4(position3, 1.0);
   vec4 vertPosWS = modelMatrix * vertPos;

   if(leftOrRight < 1){ //Left eye
      gl_ViewportIndex = 0;
      gl_Position = viewProjMatrixLeft * vertPosWS;
   }
   else{ //Right eye
      gl_ViewportIndex = 1;
      gl_Position = viewProjMatrixRight * vertPosWS;
   }
}
```

Listing 2. GLSL vertex shader illustrating stereo instancing combined with conventional geometry instancing.

also the number of occlusion tests becomes reduced by a factor of two and the culling efficiency is only marginally reduced (i.e., an object visible in either viewport will be scheduled for rendering into both).

In addition, it is straightforward to extend stereo instancing to also support conventional (dynamic) instancing as proposed in [Johansson 2013]. Listing 2 provides an example vertex shader that generalizes the use of instancing for both geometry replication as well as stereo-pair generation. For a complete picture of this approach the reader is referred to the original Johansson paper. Here, a shared array of indices (I) is used to locate each instance's transformation matrix, encoded in a single, shared array (M). Without stereo instancing enabled, the index to fetch corresponds to `gl_InstanceID` plus a per-geometry type offset (every geometry type gets a certain offset into I). With stereo instancing enabled, the instance count is increased by a factor of two, however, the composition of the index array (I) remain unchanged. First, the modulus operator (%) is used to distinguish between left and right instances, i.e., even numbers go to the left viewport and odd to the right. Second, the `gl_InstanceID+offset` is divided by two, and the integer part is used to lookup and fetch the instance's transformation matrix (the model matrix is the same for both the left and right eye). Additionally, the code in Listing 2 takes advantage of the `NV_viewport_array2` extension in order to access multiple viewports in the vertex shader, thereby removing the need for a screen-space transformation as well as custom clipping.

3.2. Batching of Non-instanced Geometry

Even with occlusion culling and the instancing-based approaches to reduce the number of draw calls, certain viewpoints may still require a large number of individual draw calls. However, many non-instanced objects in a BIM, such as walls, contain very few triangles, which make them suitable for geometry batching. In the case of wall objects, the overall low triangle count per object is a result of each wall segment—straight or curved—being considered a singular object in BIM authoring systems (i.e. Autodesk Revit or ArchiCAD). As a result, the number of wall objects represents a significant amount of the total number of objects contained in a BIM while the number of wall triangles only represents a fraction of the total amount of triangles in the model.

As BIMs contain detailed information about each individual object (i.e., metadata), it becomes trivial to collect and batch wall geometry by material during scene loading. Although more sophisticated methods could be used to create optimal sets of wall geometry, they are simply batched by material and the centroid position of the walls bounding box (i.e., spatial coherence) in order to form clusters of approximately 7500 triangles each.

4. Performance Evaluation

The different stereo rendering techniques have been implemented in a prototype BIM viewer and tested on three different BIMs taken from real-world projects (Figure 4). All three models (dormitory, hotel, and office building) were created in Autodesk Revit 2014 and represent buildings that exist today or are currently being constructed. Although the hotel model contains some structural elements, they are primarily architectural models; as such, no mechanical, electrical, or plumbing (MEP) data is present. However, all models contain furniture and other interior equipment. In Table 1, related statistics for each model are shown. Please note both the large number of instances as well as small number of triangles per batch for the wall objects, which motivates the use of hardware-accelerated instancing as well as geometry batching of wall objects.

All the tests were performed on a laptop equipped with an Intel Core i7-4860HQ CPU and an Nvidia GeForce GTX 980M GPU with Nvidia graphics driver 361.43 in-

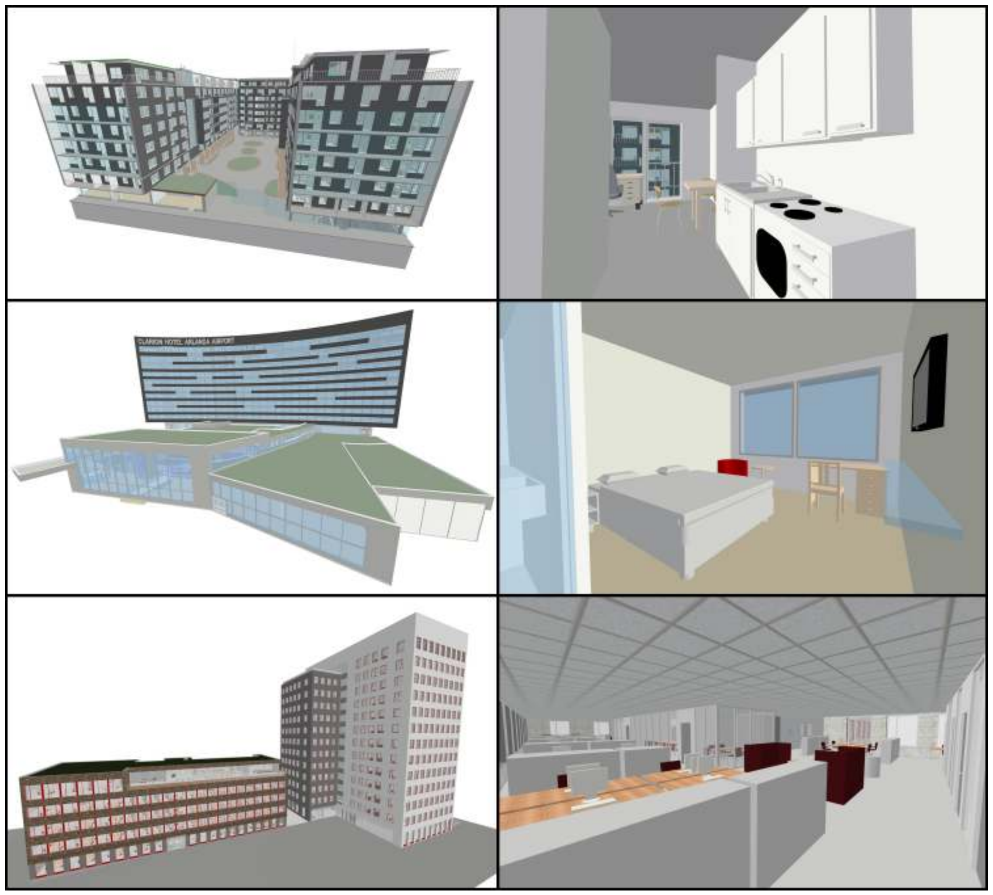

Figure 4. Exterior and interior views of the different test models. From top to bottom: dormitory, hotel, and office building.

	Dormitory	**Hotel**	**Office**
# of triangles (total)	11,737,251	7,200,901	12,454,919
# of objects (total)	17,674	41,893	18,635
# of batches (total)	34,446	60,955	26,015
# of instanced batches / total # of batches	0.60	0.56	0.81
# of wall batches / total # of batches	0.38	0.37	0.12
# of wall triangles / total # of triangles	0.014	0.04	0.004
Avg. # of triangles per wall batch	12	13	16

Table 1. Model statistics for the three different BIMs.

stalled. The resolution was set to 2364 × 1461 pixels (off-screen buffer) with 4x multisample antialiasing. All materials use a very simple N dot L diffuse color/texture shader.

Upon model loading, a bounding volume hierarchy (BVH) is constructed according to the surface area heuristics (SAH). Unless otherwise stated, the leaf nodes in this hierarchy represent the individual building components, such as doors, windows, and furniture. For each object, one vertex buffer object (VBO) is constructed per material., i.e., a typical window object will be represented by two VBOs, one for the frame geometry and one for the glass geometry.

The implementation of CHC++ is based on the description and accompanying code presented in [Bittner et al. 2009]. All proposed features of the original algorithm are used, except for the *multiqueries* and *tight bounds* optimizations. A major requirement in order to provide an efficient implementation of CHC++ is the ability to render axis-aligned bounding boxes with low overhead for the occlusion tests. Primarily based on OpenGL 2.1, the code in [Bittner et al. 2009] as well as [Johansson 2013] uses OpenGL Immediate Mode (i.e., `glBegin`/`glEnd`). In the prototype BIM viewer, a more performance-efficient *unit-cube approach* is used instead: Every bounding box in the spatial hierarchy maintains a 4 × 4 transformation matrix that represents the combined translation and scale that is needed in order to form it *from* a unit-cube (i.e., a cube with length 1 centreed in origo). During every querying phase of the CHC++ algorithm a single VBO, representing the geometry of a unit-cube, is used to render every individual bounding box. The unique *unit-cube transformation matrix* is submitted to the vertex shader as a *uniform* variable.

For all models, two different camera paths have been used: one interior at a mid-level floor in each building, and one exterior around each building. The exterior camera paths represent an orbital camera movement around each building while facing its center. As each building is completely visible throughout the whole animation sequence, it serves as a representative worst case scenario, regardless of culling strategy.

For all models and animation paths, several different culling and stereo rendering configurations have been combined and evaluated. The abbreviations used in the tables and figures should be read as follows: VFC for view frustum culling, OC for occlusion culling, HI for dynamic hardware-accelerated geometry instancing, BW for batching of wall geometry per material, TP for two-pass stereo rendering, GS for stereo duplication in the geometry shader, SI for stereo instancing, SI_NV for stereo instancing using the `NV_viewport_array2` extension, and BF for brute-force rendering. In the brute-force rendering case (BF) non-instanced geometry is batched together based on materials and instanced geometry is rendered using conventional hardware-accelerated instancing. Also, no occlusion culling is present. As correct depth ordering becomes difficult to maintain with the use of instancing, HI and BF render semi-transparent geometry using the weighted average transparency rendering technique [Bavoil and Myers 2008]. In all other cases, semi-transparent geometry is rendered after opaque objects in a back-to-front order using alpha blending.

4.1. Stereo Rendering with View Frustum Culling

In Table 2, average and maximum frame times are presented for the different camera paths when view frustum culling is combined with the different stereo rendering techniques. These results are mainly to illustrate the benefit of using stereo instancing without any additional acceleration strategies.

With only view frustum culling, these models are primarily CPU-bound due to a large number of draw calls in relation to the total number of triangle that are being

Stereo Rendering Technique (all modes use view frustum culling)	GeForce GTX 980M	
	Interior	Exterior
	Dormitory	
Two pass (TP)	14.3 / 39.5	51.8 / 68.4
Geometry shader duplication (GS)	12.8 / 26.0	33.7 / 36.7
Stereo instancing (SI)	11.0 / 21.5	27.3 / 34.9
Stereo instancing using NV extension (SI_NV)	**8.4 / 20.4**	**25.0 / 34.4**
	Hotel	
Two pass (TP)	53.0 / 82.7	91.9 / 100.8
Geometry shader duplication (GS)	29.1 / 42.0	46.3 / 49.1
Stereo instancing (SI)	29.2 / 42.2	47.1 / 49.9
Stereo instancing using NV extension (SI_NV)	**28.9 / 41.5**	**46.2 / 48.9**
	Office	
Two pass (TP)	6.4 / 17.6	39.7 / 52.8
Geometry shader duplication (GS)	7.1 / 17.7	35.8 / 36.5
Stereo instancing (SI)	6.3 / 15.1	29.0 / 30.0
Stereo instancing using NV extension (SI_NV)	**5.0 / 11.1**	**20.1 / 22.2**

Table 2. Comparison of average/maximum frame time in milliseconds for different rendering techniques, models, and camera paths. Numbers in bold represent the lowest numbers encountered for each test setup.

drawn (i.e., small batch size). Because of this, stereo instancing becomes a much more efficient approach. Focusing on the exterior paths, the average frame times are reduced by 27–52% compared to a two-pass alternative. It also becomes clear that the `NV_viewport_array2` extension is preferable, not only in terms of implementation simplicity, but also in terms of performance. Although not to such a high degree, a speed-up is also recorded for the geometry shader-based approach.

Still, even if the stereo instancing technique offers a significant performance increase compared to both a two-pass as well as geometry shader-based approach, the performance is not enough in order to provide sufficiently high frame rates, even for the interior walkthroughs. Consequently, to be able to guarantee a smooth, interactive experience for typical BIMs, additional acceleration techniques are needed.

Stereo Rendering Technique (all modes except brute force use occlusion culling)	GeForce GTX 980M	
	Interior	Exterior
	Dormitory	
Two pass	2.4 / 4.1	7.6 / 10.4
Geometry shader duplication	2.1 / 3.7	6.2 / 9.8
Stereo instancing	2.0 / 3.3	5.4 / 8.1
Stereo instancing (NV extension)	**1.9 / 3.3**	**4.6** / 7.3
Stereo instancing (NV extension) + Instancing + Batching	2.6 / 4.0	5.2 / **7.0**
Brute force	10.9 / 15.6	17.4 / 18.2
	Hotel	
Two pass	4.0 / 5.5	27.6 / 47.9
Geometry shader duplication	3.9 / 5.3	15.0 / 25.9
Stereo instancing	3.5 / 5.1	14.7 / 25.1
Stereo instancing (NV extension)	**3.1 / 4.3**	14.2 / 24.5
Stereo instancing (NV extension) + Instancing + Batching	3.3 / 4.5	**7.5 / 10.9**
Brute force	12.0 / 13.9	11.9 / 13.1
	Office	
Two pass	2.7 / 5.0	11.5 / 19.2
Geometry shader duplication	2.2 / 4.4	8.2 / 14.3
Stereo instancing	2.2 / 4.4	7.0 / 11.8
Stereo instancing (NV extension)	**2.1 / 4.0**	6.2 / 10.7
Stereo instancing (NV extension) + Instancing + Batching	2.3 / 4.2	**6.2 / 9.0**
Brute force	10.4 / 14.3	19.2 / 19.8

Table 3. Comparison of average/maximum frame time in milliseconds for different rendering techniques, models, and camera paths. Numbers in bold represent the lowest numbers encountered for each test setup.

4.2. Stereo Rendering with Occlusion Culling

In Table 3, average and maximum frame times are presented for the different camera paths when occlusion culling is combined with the different stereo rendering techniques. Focusing on the interior paths first, it becomes clear that occlusion culling alone is sufficient to provide the required frame rates. In fact, the maximum frame time recorded for any of the interior paths is never above 6 ms, regardless of stereo rendering technique. As these models features fairly occluded interior regions, the CHC++ algorithm can efficiently reduce the amount of geometry that needs to be rendered.

Still, when inspecting these numbers in more detail, it can be seen that stereo instancing is able to reduce both average as well as maximum frame times between 20–23% compared to a two-pass approach and 5–21% compared to the geometry shader-based approach. When occlusion culling and stereo instancing are further complemented with batching of walls (BW) and geometry instancing (HI), the performance results do not show the same consistency. Although the maximum frame times are consistently below that of the two-pass approach, stereo instancing is outperformed by geometry shader-based duplication for the dormitory model. Nevertheless, in this context, it is also important to note that it is the batching of walls—and not the geometry instancing technique—that is the main cause of the increased rendering time. In fact, the combination of occlusion culling, stereo instancing, and geometry instancing give equal or actually slightly better performance compared to only using occlusion culling and stereo instancing for the interior paths. However, batching of wall geometry was primarily introduced in order to reduce the number of draw calls for viewpoints when many objects are visible. As such it does not become equally beneficial for interior viewpoints.

In order to provide a better understanding of the performance characteristics, Figure 5 presents frame timings for the interior paths for the dormitory and for the office building. Most notably, these graphs show the huge difference in performance compared to a brute-force alternative (BF), regardless of stereo rendering technique. Although the brute-force approach is able to deliver frame times below 11.1 ms during parts of the interior camera paths, this level of interactivity cannot be guaranteed.

Focusing instead on the exterior paths in Table 3, the gain of the combined method becomes even more apparent. With all the techniques activated (OC+SI_NV+BW+HI), the average frame times are reduced by 32–73% compared to the two-pass approach and 16–50% compared to geometry shader-based duplication. To give a better overview of the performance behavior, Figure 6 presents frame timings for the exterior camera paths for the three different models. Here it can be seen how the different techniques complement each other depending on which model is rendered. For the dormitory and office building models, stereo instancing provides distinct performance gains compared to both two-pass rendering as well as the geometry shader-based ap-

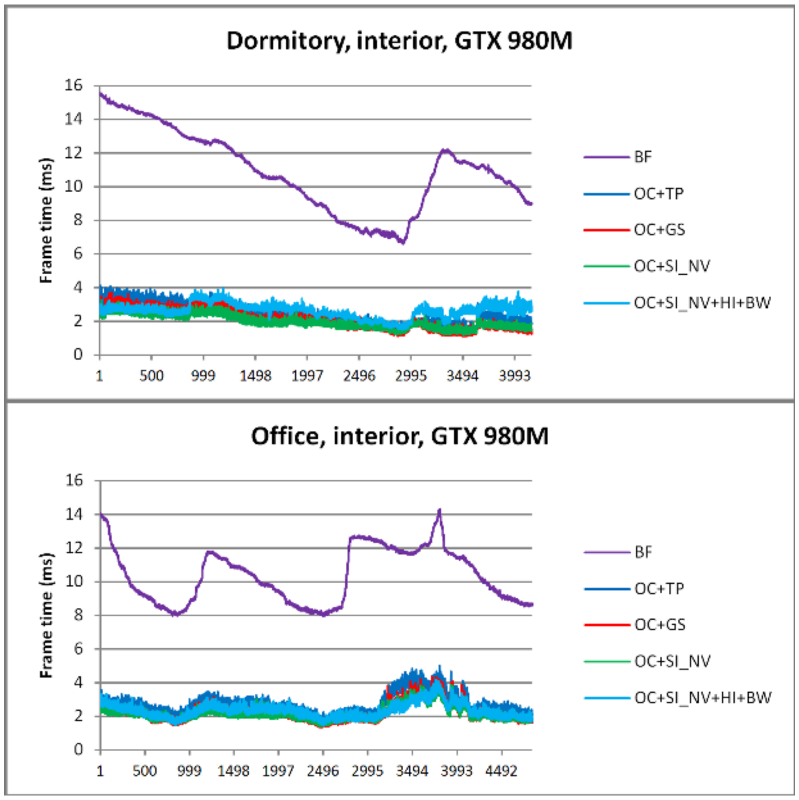

Figure 5. Frame times (ms) for the interior path of the dormitory (top) and office (bottom) buildings (BF=Brute force, OC=Occlusion culling, TP=Two-pass stereo rendering, GS=Geometry shader stereo duplication, SI_NV=Stereo instancing using NV extension, HI=Geometry instancing, BW=Batching of wall geometry).

proach. However, for these models, there is little additional to gain from conventional geometry instancing, and batching of walls. In fact, due to a high level of occlusion also in the exterior views of the student model, the average frame times are actually lower *without* geometry instancing and batching of walls activated. For the hotel model, on the other hand, the behavior is somewhat reversed. Although stereo instancing provides huge performance gains compared to two-pass rendering, the frame times are essentially equal to that of the geometry shader-based technique. Even with occlusion culling, the exterior views of the hotel model contain a huge number of visible objects and, therefore, become CPU-bound due to the large number of individual draw calls. Both stereo instancing as well as geometry shader-based duplication only address this to a certain degree. In order to reach the target frame rate, the number of draw calls has to be further reduced, which is exactly what both batching of walls and geometry instancing does. To better illustrate the individual effects of geometry instancing (HI) and batching of walls (BW), Figure 6 (middle) also presents the frame timings when these two techniques are added separately to occlusion culling

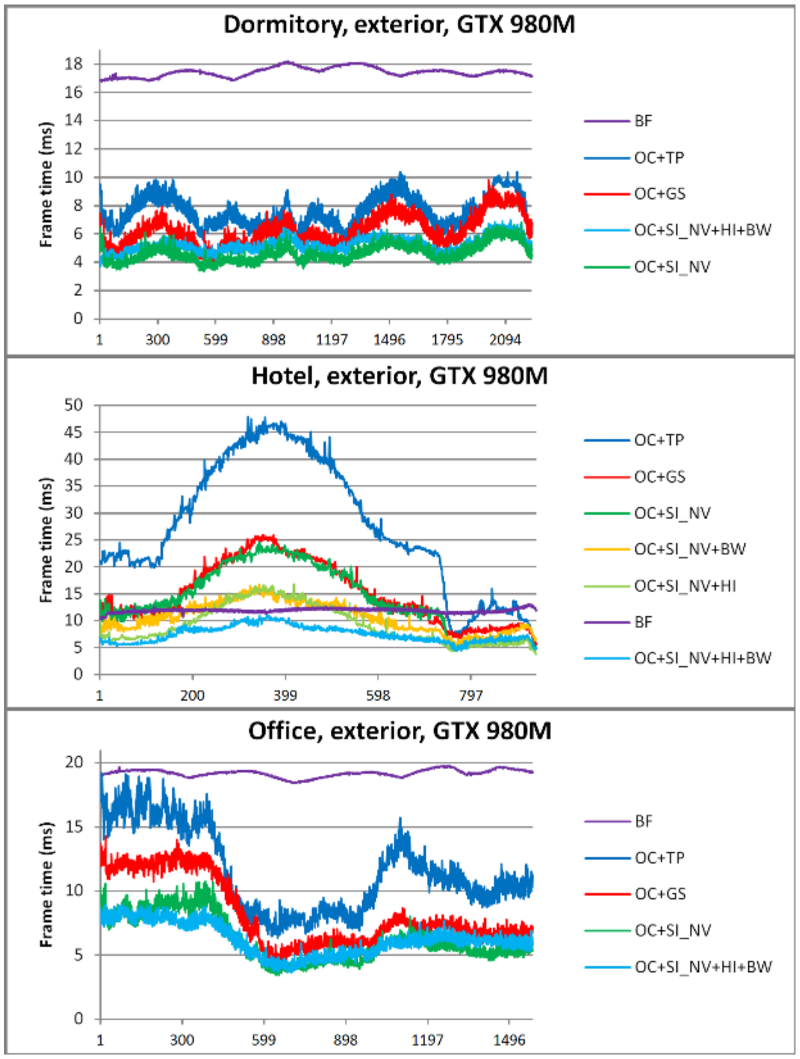

Figure 6. Frame times (ms) for the exterior path of the dormitory (top), hotel (middle) and office (bottom) buildings (BF=Brute force, OC=Occlusion culling, TP=Two-pass stereo rendering, GS=Geometry shader stereo duplication, SI_NV=Stereo instancing using NV extension, HI=Geometry instancing, BW=Batching of wall geometry)

and stereo instancing (i.e., OC+SI_NV+BW and OC+SI_NV+HI, respectively). It can thus be seen that the combination of these two techniques is required in order to reach a target frame rate of 90 Hz for the hotel model.

Another interesting aspect of the technique is that the brute-force approach is actually surprisingly close in delivering sufficient rendering performance for the hotel model. However, as seen from the results from the other two models, the brute-force alternative is clearly not scalable, especially if we also consider more complex pixel shaders. In comparison, the proposed combination of query-based occlusion culling, stereo instancing, batching of walls, and geometry instancing is able to provide the required level of performance for all of the tested models.

5. Conclusion

In this paper the stereo instancing rendering technique has been described and further investigated. The technique is very well suited for integration with occlusion query-based occlusion culling as well as conventional geometry instancing and has been shown to outperform traditional two pass stereo rendering approach, geometry shader-based stereo duplication, as well as brute-force stereo rendering of typical BIMs on recent hardware. Although occlusion culling alone turned out to provide sufficient rendering performance for interior rendering of typical BIMs, the combination of techniques proved vital in order to guarantee required frame rates also for exterior viewpoints.

References

BAVOIL, L., AND MYERS, K. 2008. Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK*, 1–12. URL: http://developer.download.nvidia.com/SDK/10/opengl/src/dual_depth_peeling/doc/DualDepthPeeling.pdf. 11

BITTNER, J., MATTAUSCH, O., AND WIMMER, M. 2009. Game-engine-friendly occlusion culling. In *SHADERX7: Advanced Rendering Techniques*, W. Engel, Ed., vol. 7. Charles River Media, Boston, MA, Mar., ch. 8.3, 637–653. URL: http://www.cg.tuwien.ac.at/research/publications/2009/BITTNER-2009-GEFOC/. 10

HILLAIRE, S. 2012. Improving performance by reducing calls to the driver. In *OpenGL Insights*, P. Cozzi and C. Riccio, Eds. A K Peters/CRC Press 2012, Natick, MA, 353–364. URL: http://www.crcnetbase.com/doi/abs/10.1201/b12288-30, doi:10.1201/b12288-30. 3

JOHANSSON, M., ROUPÉ, M., AND BOSCH-SIJTSEMA, P. 2015. Real-time visualization of building information models (BIM). *Automation in Construction 54*, 69–82. URL: http://dx.doi.org/10.1016/J.AUTCON.2015.03.018, doi:10.1016/j.autcon.2015.03.018. 2

JOHANSSON, M. 2013. Integrating occlusion culling and hardware instancing for efficient real-time rendering of building information models. In *GRAPP 2013: Proceedings of the International Conference on Computer Graphics Theory and Applications, Barcelona, Spain, 21-24 February, 2013.*, SciTePress, Lisbon, 197–206. URL: http://publications.lib.chalmers.se/records/fulltext/173349/local_173349.pdf, doi:10.5220/0004302801970206. 2, 6, 8, 10

MARBACH, J. 2009. Gpu acceleration of stereoscopic and multi-view rendering for virtual reality applications. In *Proceedings of the 16th ACM Symposium on Virtual Reality*

Software and Technology, ACM, 103–110. URL: http://dl.acm.org/citation.cfm?id=1643953, doi:10.1145/1643928.1643953. 3

MATTAUSCH, O., BITTNER, J., AND WIMMER, M. 2008. Chc++: Coherent hierarchical culling revisited. *Computer Graphics Forum 27*, 2, 221–230. URL: http://dx.doi.org/10.1111/j.1467-8659.2008.01119.x, doi:10.1111/j.1467-8659.2008.01119.x. 6

TRAPP, M., AND DÖLLNER, J. 2010. Interactive Rendering to View-Dependent Texture-Atlases. In *Eurographics 2010 - Short Papers*, The Eurographics Association, Aire-la-Ville, Switzerland. URL: http://dx.doi.org/10.2312/EGSH.20101053, doi:10.2312/egsh.20101053. 4

VLACHOS, A. 2015. Advanced VR rendering. Presentation at Game Developers Conference 2015. URL: http://alex.vlachos.com/graphics/Alex_Vlachos_Advanced_VR_Rendering_GDC2015.pdf. 2

WILSON, T. 2015. High performance stereo rendering for VR. Presentation at San Diego Virtual Reality Meetup. URL: https://docs.google.com/presentation/d/19x9XDjUvkW_9gsfsMQzt3hZbRNziVsoCEHOn4AercAc. 2

Author Contact Information

Mikael Johansson
Department of Civil and Environmental Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
jomi@chalmers.se

Mikael Johansson, Efficient Stereoscopic Rendering of Building Information Models (BIM), *Journal of Computer Graphics Techniques (JCGT)*, vol. 5, no. 3, 1–17, 2016
http://jcgt.org/published/0005/03/01/

Received:	2015-05-06		
Recommended:	2016-03-02	Corresponding Editor:	Warren Hunt
Published:	2016-07-25	Editor-in-Chief:	Marc Olano