Large Scale Geometric Visibility Culling on Brave
Pixar Technical Memo #13-05

Zachary Repasky* Patrick Schork Kevin McNamara Susan Fong
Pixar Animation Studios

Disney·Pixar's *Brave* is visually complex, containing fully clothed and articulated crowds characters, a forest full of vegetation, ruins littered with debris, and a full fledged castle. In fact, it is so complex that our previous methods to remove unnecessary geometry are no longer adequate in keeping the renders feasible. On prior shows, we used a low quality render to determine an object's visibility per shot. However, even a shot-level removal of the geometry limited the artist's turnaround times. To address this, we developed a new two pass algorithm that aggressively culls geometry on a per frame-level, while maintaining accurate visibility. The approach first culls non-visible geometry in the shot using low quality renders on sparse frame intervals containing the full data set. The culled set is then further refined on a per frame-level using a heuristic based on object to camera relationships. In addition to our implementation that uses a software renderer, we developed a GPU version that allows immediate geometry removal.

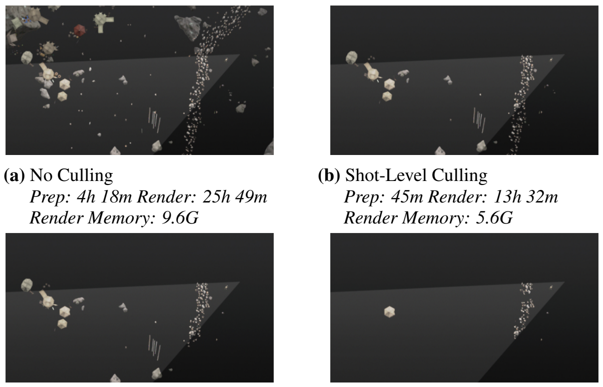
1 Basic Algorithm

Brave's *sets* are created as *full environments* based on location. Many shots are placed in these locations and therefore each shot contains an entire environment worth of data. The complexity of the data demanded aggressive culling. Interactive approaches [Hudson et al. 1997] could be used to quickly approximate culling using large data sets, but these approaches don't provide accuracy when using displacements, transparencies, and dynamically loaded procedural geometry.

Shot Based Visibility Culling Pass: Before the visibility culling passes, each shot includes the full set. Since off-screen objects can include illumination dependencies such as reflections and shadows, we use a camera with a field-of-view (FOV) offset greater than the shot camera, and limit culling to a radius around its location to minimize potential visual differences. To shrink the data per shot, renders with basic lighting are run on sparse frame intervals and employ an atmospheric shader to determine the visible objects per pixel. Each pixel has the potential to store multiple objects at varying depths to account for material transparencies. The raw output for each frame gets processed and written to a cache containing only visible objects. After the renders complete, each cached frame is loaded and combined into one data set that contains all shot-level cullable objects. Next, an exclusion pass runs to avoid removing objects or groups that the user defines as non-cullable, such as a shadow casting tree that was missed by the culling camera. To complete the first pass we do a visible object constraint analysis on the refined set to determine if any objects in the cullable set are needed for computation at final render-time by the visible objects.

An optional secondary feature culls away geometry that was visible onscreen, but made little visible contribution to the final image. This was helpful on large establishing shots where the majority of the set was in camera. To do this we used image heuristics based on the raw cache output from the render to determine the percentage of screen space each object occupies. If a visible object is found with visible area below a specified threshold, it is marked cullable and further checked for dependencies as discussed in the above paragraph. Once the pass is completed, pre and post renders are verified to ensure acceptable visibility before authoring the results to a file.

*zkr,pschork,sfong@pixar.com



(a) No Culling
Prep: 4h 18m Render: 25h 49m
Render Memory: 9.6G

(b) Shot-Level Culling
Prep: 45m Render: 13h 32m
Render Memory: 5.6G

(c) Frame-Level Frustum Culling
Prep: 32m Render: 13h 30m
Render Memory: 5.6G

(d) Frame-Level GPU Culling
Prep: 15m Render: 9h 45m
Render Memory: 4.8G

Figure 5: *This is an example of an actual production shot. The non-visible items vastly dissapear when using the frame based approaches. Notice in figure (d), occluded geometry is removed by forground elements. © Disney·Pixar*

Frame Based Visibility Culling Pass: Since the geometry set is so complex, even after the shot-level pass, a pre render process was developed to cull a frame on the fly, immediately before passing it to the renderer. Two approaches were developed to achieve this, an approximate frustum culling based approach, and a more accurate screen space GPU accelerated approach. The camera frustum centric approach gathered bounding boxes for each model and evaluated them against the camera frustum to determine approximate frame-level visibility.

Since the frustum centric approach only culled what was outside of the camera, a GPU accelerated method was developed to account for object occlusion. To achieve this we build an image histogram to represent the distribution of screen space pixels to scene nodes. Hierarchical sampled scene data is loaded into a custom vertex buffer object (VBO) renderer. The renderer assigns each node a unique RGBA value. An image is rendered to an off-screen framebuffer using GL select mode. After the histogram has been built from the select image, nodes with pixel counts below a specified threshold can be easily queried for and culled. If transparency needs to be taken into account, a multi pass approach, where semi opaque nodes are iteratively hidden, can be used.

As in the shot-level pass, a dependency analysis is executed using either approach to check for geometry whose locations are needed to evaluate visible objects at render-time.

The two passes provided a 80-90% render preparation time decrease and lowered render memory use by 40% or more, allowing the film to be feasibly completed. We thank Yasmin Khan, Gordon Cameron, Ariela Nurko, Tim Babb, Brett Warne, Max Planck, Paul Kanyuk, Don Schreiter, Danny Nahmias, Arun Rao, and David Yu.

References

HUDSON, T., MANOCHA, D., COHEN, J., LIN, M., HOFF, K., AND ZHANG, H. 1997. Accelerated Occlusion Culling using Shadow Frusta. In *Proc. on Comp. Geometry 1997.*, 1–10.