CS 557 -- Winter Quarter 2021

Paper Analysis Project

Oregon State University

Qingxiao Yuan

The theme of the paper and the meaning

The paper I read is "Computing shortest PATH maps with GPU shaders."(Camporesi, 2014). The authors introduced a shortest path map (SPM) method, which is based on the GPU's new calculation method for the source points in the polygon area. They took the advantages of GPU polygon rasterization to use shader programming. After encoding the SPM buffer in the frame, the global shortest path can be efficiently calculated in time, which is proportional to the number of vertices in the path, and the length query is calculated in a constant time. The meaning of the title is that using the GPU shaders to calculate the shortest path map. They are trying to compute the Shortest Path Maps with GPU Shaders in a relatively faster way. Although SPM is well known in computational geometry, they did this because it is still little known in other fields of computer graphics, and it has not been extensively studied in practical applications. One of the reasons for this is that effective methods for calculating SPM are not easy to implement, which proves the need to develop GPU-based alternative methods. The results of various environmental assessments show that the speed has been significantly improved. The authors first rely on the standard CPU algorithm to calculate the shortest path tree of the obstacle set, and then use the proposed shader to encode the SPM in the frame buffer with arbitrary resolution. Finally, they used GPUs to do the operations. I think they used this method because the GPU could do a bunch of work effectively.

Authors' background

Both Carlo Camporesi and Marcelo Kallmann came from University of California, Merced.

"Carlo Camporesi is a PhD candidate at the University of California Merced. His interests are 3D visualization, virtual reality, computer animation and simulation. His current research focuses on the development of new methods for motion tracking, interactive controls, and the use of 3D visualization technology for collaborative virtual environments and human-computer interaction. He joined UC Merced's graphics laboratory in 2008 to build UCM virtual reality facilities. Before that, he was a researcher at the Italian National Research Council (ITABC), where he developed an interactive visualization application for the virtual heritage project, focusing on the transmission of 3D datasets over the network. He graduated in Computer Science (MS) in 2005, and his thesis topic is generating large-scale archaeological landscapes (from GIS datasets) via the Internet. This work was developed in cooperation with CINECA, Italy's largest non-profit computing center." (Ucmerced)

"Marcelo Kallmann is Professor and Founding Faculty at the University of California, Merced. He was chair of the EECS graduate program from July 2018 to June 2020. He holds a PhD from the Swiss Federal Institute of Technology in Lausanne (EPFL), was research faculty at the University of Southern California (USC), and a research scientist at the USC Institute for Creative Technologies (ICT), before moving to UC Merced in 2005. A significant part of his research interests are on motion planning algorithms for a wide range of applications in robotics and computer graphics. He has served as program co-chair for the 5th International Conference on Motion in Games (MIG) in 2012 and as Associate Editor for ICRA in multiple years. He is also an Associate Editor for the Computer Animation and Virtual Worlds journal. At UC Merced he established the computer graphics group (http://graphics.ucmerced.edu/). He is recipient of several NSF awards and his recent work on triangulations for path planning runs inside The Sims 4, the best selling PC game of 2014 and 2015." (Ucmerced)

The experiments they did

The first step of their algorithm is to generate the extruded volume of the input polygon along the z-axis of the environment, which is the axis orthogonal to the plane of the polygon domain. The generated visibility graph follows the standard method of connecting all pairs of vertices that are visible to each other, and is optimized to include only the edges connecting two convex vertices. Their implementation determines the visible vertices by traversing the triangulation of the environment outward from each vertex. This method takes O(n^3) time, but in fact it effectively eliminates invisible areas of the environment. Each node of the visibility graph stores a list containing the IDs of all visible obstacles. Since they were all concerned about the algorithm problem, they used O to measure the execution time.

Then, every time a new source point is given, the point is first connected to the visible node of the visibility graph through an edge. Then, an exhaustive Dijkstra expansion is performed from the newly inserted source node, and the resulting expanded tree is the SPT of the polygonal environment. Each node of SPT will correspond to a specific obstacle vertex. In the SPT construction process, each vertex is marked to identify whether (part of or all) the edges connected to the vertex are visible from its SPT parent vertex (or one of the vertex is completely occluded). The SPT nodes adjacent to the visible edge will be deleted from the SPT because they will not be generators of the SPM area. The remaining nodes will be generators and will be assigned a unique color ID. The generator nodes are also stored in a list sorted by the Euclidean geodesic distance of each node to the source point.

Their method of obtaining SPM was inspired by the GPU rendering cone method, which is a method of expressing the distance to the boundary, and was originally proposed for calculating the generalized Voronoi diagram. The height of each cone is equal to its radius. In the case of the source point, no need to trim the cone. When rendering the cone from a front view, the resulting depth buffer will contain the distance to the generator point.

They solved the problem by generating a trimmed cone for each vertex to ensure that each cone propagation does not affect the area that is not visible from the generator vertex of the cone. However, a CPU calculation of a specific shape must be performed for each vertex, and for each new SPM source, it must be calculated and passed to the graphics hardware. This is also where they would take the advantage of the graphics hardware.

They used the concept of point light propagation and shadows, in which each SPT node is an omnidirectional light source. Hence, they are directly generated in the GPU shadow volume to identify which specific cones are during each cone rendering Partially applied to the rendering, and thus only used to update the cone. The relevant color and depth buffer values. This method relies on the sequential update of the buffer and requires the cone rendering to follow a strict order, which is given by the sorted geodesic distance to the SPM source.

They used two shader programs. The first shader technology (techFlat) can simply render the cone with a specified color without applying any shading information from the geometry. The second shader program (techShad) is responsible for calculating the amount of shadow that will project the geometric contour of the environment to infinity (the projection point is the SPT generator node or the SPM source point). This shader is a modified version of the most common contour generation algorithm. The algorithm proposes a general method to create shadows from any angle using geometric figures with adjacency information. They also developed a dedicated version from the perspective of an orthographic camera, using only general geometry (not containing adjacency information). In order to approximate the vertex projection to infinity, the shader uses a simple solution that uses a value very close to 0 as the uniform component of w in the normalized direction defined by the light source and the vertex.

The conclusion of the paper

The authors have implemented shaders in OpenGL 3.3 or higher and have used C++ in the CPU part. For the frame buffer object rendering process, they used 32-bit RGBA rendering targets (used to store colors) and 24-bit depth and 8-bit template rendering targets (storing depth and template information respectively). The authors tested general environments with complex obstacles. Compared with previous results, the authors' method shows a 20-fold increase in speed with the same number of obstacles, map resolution, and a slightly weaker graphics card (480 computing cores instead of 512 cores). After refreshing the graphics card rendering pipeline with glFinish, the calculation will be performed every time. Under normal settings, this command is not necessary, and the rendering time of the query will be shortened. The authors' SPM indicates that it supports constant-time distance query of any discrete point in the SPM, and can return a path proportional to the turning time of the path. In addition, the authors also developed a shader that can use color map representation to simulate continuous geodesic flood fills the expansion. This process replaces the TechFlat shader with a geodesic flood fill program, which takes into account the shortest path from the pixel to the source point, and converts the geodesic distance of each pixel into a specific color value. The shader uses HSV color notation to evenly map distances to colors, and adds white space to highlight the boundaries of the propagating waves. They also introduced a novel method that can use GPU shader programming to generate discrete shortest path graphs from the polygon domain. The proposed algorithm has been shown to be more than 20 times faster than previously reported results. This improvement can generate more complex results and obtain SPM images with complexity and detail not shown in previous work. I think their results contribute to a deeper understanding of SPM and its applications.

New insights for me

The digital precision of the graphics card is passed to the shader. The light source position is usually the SPT node (so it happens to be at the vertex of the polygon), and due to accuracy errors, the graphics card may mistakenly think that it is inside the polygon, in which case the result may be incorrect. This problem can be avoided by placing the source light slightly outside the obstacle along the bisector vector between the two adjacent edges of the vertex. I think here they apply the bump-mapping we used in project3-Displacement Mapping, Bump Mapping, and Lighting. Since bump-mapping also achieves visual "realism" by manipulating the position of the light.

The proposed algorithm for Z buffer accuracy depends on the accuracy of the depth buffer. They considered the angle of the camera to appropriately set the zNear value so that the entire scene can be rendered correctly when rendering the scene from a common angle for debugging. I think here they used our sharpening function in project5-Image Manipulation in a "Magic Lens"- to improve accuracy through sharpening.

Z-fighting Since each cone mesh has the same slope, and the cones expanded by the children are placed exactly at the boundary of the parent, polygons belonging to different cones but with child relations may produce Z-fighting artifacts. To avoid this problem, they added a small gap in the height of the child's cone. This gap ensures that the trimmed conical mesh is always rendered below the trimmed conical mesh of the parent object. Since this value depends on the accuracy of the depth buffer, adaptive calculations can also be performed. I personally think that this is a bit similar to the smoothstep() in the mixing we learned, or the function used.

Flaws in the paper's methods and conclusion

Regarding the vulnerability of the author's article, I think we can start from the universality of graphics cards. Because the author did not mention the differences in graphics cards between different manufacturers. I don't know if that difference will cause them to use graphics cards from different manufacturers. As a result, the efficiency may also be inconsistent when using GPU computing.

The next step for me about this research

Regarding how to improve, I agree with the author's point of view. First, I think many improvements to the proposed shader program can be achieved, and many extensions to the proposed algorithm can be made. For example, pixel buffer objects instead of frame buffer objects may result in faster non-blocking SPM queries, and template-based 2D shadow projection using quads extruded from visible edges instead of full 3D shadows may also produce more Quick results.

Based on the vulnerabilities that may be caused by different graphics cards, I think it is possible to test the author's method in this article from different graphics cards. If the graphics card display is very different, then another research direction will come out. If the graphics card display is not much different, or it is almost negligible, the method of this article can be made more robust and universal.

References

Camporesi, C., & Kallmann, M. (2014). Computing shortest PATH maps with GPU shaders. *Proceedings of the Seventh International Conference on Motion in Games*. doi:10.1145/2668064.2668092

Ucmerced. https://ucmerced.academia.edu/CarloCamporesi

Ucmerced. http://graphics.ucmerced.edu/~mkallmann/