

That's a wrap: Manifold Garden rendering retrospective

Arthur Brussee
arthur.brussee@gmail.com
William Chyr Studio

Andrew Saraev
William Chyr Studio

William Chyr
William Chyr Studio

CCS CONCEPTS

• **Computing methodologies** → **Non-photorealistic rendering**; • **Applied computing** → **Computer games**.

KEYWORDS

games, stylized-rendering, edge shading, toroidal geometry

ACM Reference Format:

Arthur Brussee, Andrew Saraev, and William Chyr. 2020. That's a wrap: Manifold Garden rendering retrospective. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '20 Talks)*, August 17, 2020. ACM, New York, NY, USA, 2 pages. <https://doi.org/10.1145/3388767.3407385>

Manifold Garden is a stylized, Escher-esque puzzle game. It was quite a unique project; there was a lot of freedom to explore graphics tech for the game. The end result was a tighter co-evolution of art and tech. This talk will describe some solutions we developed in depth, and present some of our overall takeaways.

1 TOROIDAL GEOMETRY RENDERING

The world of Manifold Garden consists of an infinitely repeating world. That is - if you fall off the bottom of the world you'll end up on top again. We relied a sleight of hand for this effect: Duplicate the world in a grid, and teleport the player when they reach the boundary of the centre world instance. This naive approach worked surprisingly well, as long as one takes care to make heavy use of Instancing, and LODs. We created an automatic decimation pipeline that created LODs for further away wrap instances, and kept the art workflow uninterrupted. For dynamic objects, this approach was still too slow; there are n^3 instances to update in the world. Instead, for these we relied on a fully GPU driven approach:

- Write out transform data for the centre instance to a compute buffer
- Frustum cull each instance, in each cell, in a compute shader; append visible instances to a buffer.
- Use indirect dispatch to render all instances in one pass.

This approach saved us from having to update separate game entities. In the future, this technique could be scaled up and used for everything, but it currently remained with too many limitations regarding sorting and LODs to be used for everything. Another interesting point to consider are shadows. Of course, in a real toroidal space light propagation is quite bizarre! Rather we pretend light is

propagated from infinity and blocked only by a fixed number of wrapped instances. This requires very large shadow maps to work. To increase our apparent shadow map resolution, we used two approaches: Firstly we used an adaptive threshold of a filtered shadow map. Secondly, we used a 'cabinet' projection for our shadow map, that is, we rotate the shadow camera basis such that two axes of the light are axis-aligned. For geometry that is axis-aligned this greatly reduced aliasing.

2 PORTALS

Levels are linked through 'portals' which act like wormholes in the world. To create this effect, it was important to define a clear data flow for our rendering pipeline. Each portal renders recursively; this rendering is too complicated and stateful to do as an actual recursive function. Instead, each render pass is passed an explicit stack that contains the current rendering setup. This allowed us to define a clear dataflow for each rendering pass. It's often overlooked, but it's where our code base really helped. The render loop is a tight, data-oriented stack; a loosely coupled, stateful, Object-Oriented style stack without a clear grip on the underlying data would have failed. When each portal renders, we use the stencil buffer to determine the portal visibility. For each portal we stamp a part of the screen with its portal ID. This is done with a two pass stencil approach: First we apply a 'testing' pass that compares to the current portal ID, and if equal writes a 1 to the highest stencil bit. The second pass writes the new portal ID to the low 7 bits where the high bit is 1. This allows us to support arbitrary setups of portal recursions. It's tempting to use e.g. stencil increment operators and write portal IDs directly - but this will always have edge cases. The two pass approach allows for arbitrarily complicated setups.

3 EDGES

Manifold Garden's aesthetic relies heavily on its painterly shaded edges. We render these as a post processing effect, aided by an extra buffer containing normals & depth metadata. First, consider a naive edge implementation [Decaudin 2009]: Check if any 4-neighbor pixel (up, down, left, right) has a 'different' normal or depth; if so, this pixel is an outline! Unfortunately this aliases terribly. It effectively renders outlines at a quarter resolution: if $A \rightarrow B$ is an edge so is $B \rightarrow A$. Additionally, edges have no defined width meaning the theoretical nyquist frequency is infinitely large. To fix this, first let's create a good definition of when pixels are considered 'different'. First, to compare normals, we use a threshold on the maximum angle between the world-space normals ($\vec{n}_{\text{neighbour}} \cdot \vec{n}_{\text{central}} < 1 - \cos[\alpha_{\text{threshold}}]$). To compare depths, we use a threshold on the maximum worldspace distance of a pixel, to the plane represented by the normal and position of the central pixel ($|\vec{n}_{\text{central}} \cdot (\vec{r}_{\text{neighbour}} - \vec{r}_{\text{central}})| > d_{\text{threshold}}$). This definition takes into account the first-order curvature of the geometry, and is much easier to tweak than a naive straight threshold approach.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGGRAPH '20 Talks, August 17, 2020, Virtual Event, USA

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7971-7/20/08.

<https://doi.org/10.1145/3388767.3407385>

It solves cases of false edges at high incidence angles, and could be extended to second order curvature if needed. Secondly, let's consider a better method to shade an edge: If two pixels differ, there is an edge somewhere between them, meaning every pixel has some distance to its nearest edge. This means we effectively define a signed distance field (SDF). We can now consider the edge shading equivalent to rendering anti-aliased iso-lines of this SDF using well known techniques [Green 2007]. Most importantly, this definition extends to arbitrary sample positions instead of just grids. Manifold Gardens supports MSAA, and using this definition we can calculate the SDF for each MSAA subsample by testing for an edge against every other subsample, effectively sampling the SDF at the full MSAA resolution.

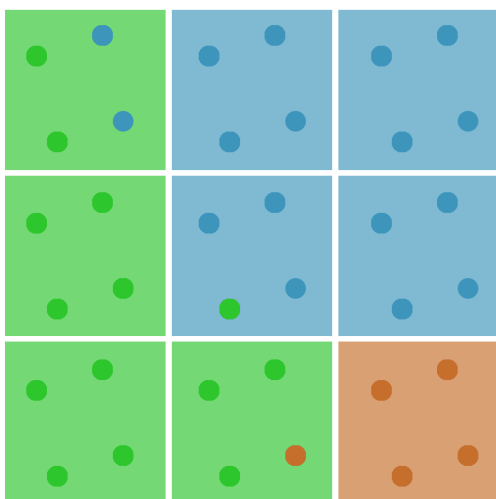


Figure 1: Example of the complication of edge detection with MSAA. Each subsample can potentially have an edge between it and another subsample. We calculate the minimum distance for each subsample and render the resulting SDF.

The performance of this is not great however. Two further observations make this approach scale:

- Most pixels are not edges. We can do a simpler broad check first, to see if a pixel can be an edge, and if not, skip the slow path.
- Edges are symmetric, and we can prove the coarse pass only needs to be done at a $1/4$ resolution!

4 PERFORMANCE CONFIGURATION

Late in Manifold Gardens production we decided to also ship on iOS. The game was running smoothly on a PS4, but that is a far cry from running well on an iPhone 7. We first followed the tried and true recipes; crunch down our geometry, speed up shaders, speed up our post-processing, etc. but nothing was quite going to get us there. As such, we had to create a flexible rendering pipeline with various knobs to turn to trade fidelity for rendering cost. We created

'performance profiles', groups of settings that land somewhere in this fidelity/cost space. Of course every game has quality settings, but this clearer fidelity/cost space framing allowed us to pick out these points in a data driven way. We profiled various groups of settings, using a performance test sweeping all of our scenes, to make sure they were fixed performance increments apart, trading off increments of 'fidelity' as judged by us. We shipped this same system on other platforms to aid users selecting the right settings; it's not reasonable to ask users to configure some high dimensional space of settings to find their preferred point in the performance / fidelity space.

Render Cost vs. Fidelity

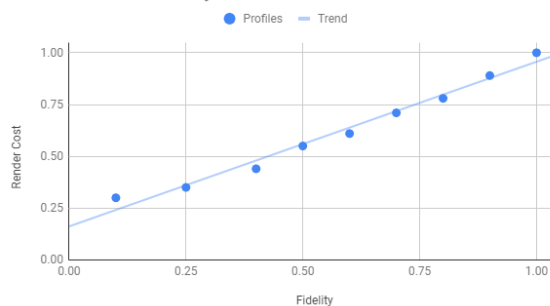


Figure 2: Rather than arbitrary quality settings, we created a data driven pipeline to determine good settings in some cost/fidelity space.

5 MAIN TAKEAWAYS

We presented techniques showing some of our approaches to clean stylized rendering. These are mostly not directly applicable to most games, but the spirit of all of them can be. Keeping rendering pipelines data driven can save immense headaches. It allowed us to effectively implement effects like our portals. Explicit handling of MSAA samples can be very worthwhile in many cases. Specialized shadow techniques are worth exploring. Thinking clearly about your render pipeline configuration can allow for greater flexibility.

REFERENCES

- Philippe Decaudin. 2009. Cartoon-Looking Rendering of 3D-Scenes. *Research Report INRIA 2919* (2009).
- Chris Green. 2007. Improved Alpha-Tested Magnification for Vector Textures and Special Effects. (2007).