A Representation of Renderers

Nicolas Feltman

November 28, 2012

1 Definitions

1.1 Entities

Our representation uses a point-free style, meaning that the entities that our algorithm is processing are not explicitly represented with variables. Nonetheless, those entities do exist, and to understand our representation one must first understand the possible types of those entities. Those are:

- **GS-Tuple** (description)
- **Hits** (description)
- **Fragment** (description)

1.2 Nodes

In this section, we present all of the node building blocks from which our representation is built.

The first things to cover are the transitional nodes, namely Hit and Shade. These are the only nodes that allows entity type change, and so they must be present in order for the system to do any work. The transition nodes are nullary, meaning that they don't require any child nodes. The Shade node always takes in a single Hit entity, and outputs a Fragment entity with the same key. The hit entity takes in a GS tuple containing a single geometry element and a single sample element, and returns their intersection point if and only if that intersection exists. If no intersection exists, then the Hit node has no output.

The decomposition nodes (*eg.* 1G, 1S, 1H, 16^2-SP, 2-GP), also nullary, tell the algorithm how to divide the entities being operated on at that moment. For instance, 2-GP indicates that the algorithm should split the current GS-tuple into two others, based on the geometry. That is, half the geometry goes to one GS tuple, half goes to the other, and all samples go to both tuples. Similarly, 16^2-SP means that the samples should be divided into 256 bins, using some 16×16 grid, and that all geometry should go to each bin. The 1G and 1S nodes indicate that the algorithm should split up the geometry and samples into their individual units, but still paired with the full set of the other. 1H similarly indicates to break up the hits fully. All of these decompositional nodes can be thought of as having one input and many outputs.

The filter node, which is nullary, allows a renderer to discard GS-tuples which will not produce any hits. They do this by by forming bounds around all of the geometry and all of the samples,

according to the bounding arguments passed in. If these bounds have any intersection, then the whole GS-tuple passes, otherwise it is dropped. Thus the filter node can be thought of as having one input, and zero or one output.

The chain node, $\ggg$, is our first node with children. Specifically, it is binary. We can interpret $\ggg$ to mean "do the thing on the left, then do the thing on the right," where the child on the left has no more than one output (*eg.* a transition node or a filter). The number of outputs of $\ggg$ is equal to the number of outputs of the child on the right. Of course, if the item on the left ends up having no outputs, then the item on the right never executes, and the $\ggg$ itself has no outputs.

The $\oplus$ and $\cup$ nodes are binary combinator nodes, like $\ggg$, but they permit children on the left with multiple outputs. The item on the right must have zero or one outputs. Thus $\oplus$ and $\cup$ operate by running the left child on the current input, and then for all outputs running the right child. These outputs are then merged together in a way unique to $\oplus$ and $\cup$. For this reason, these two are called merge nodes. In particular, $\oplus$ merges by comparing keys and taking closer hits for those keys.

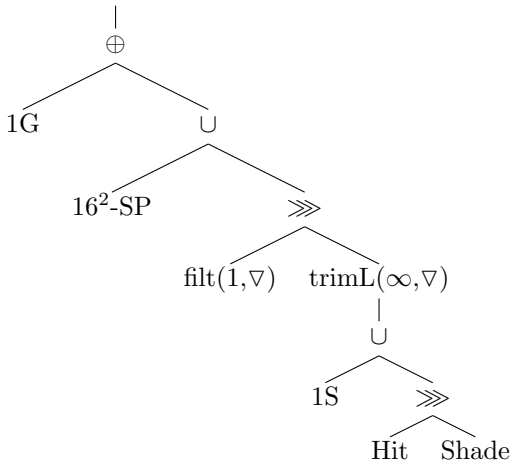
1.3 Bounds

Cover bounding.

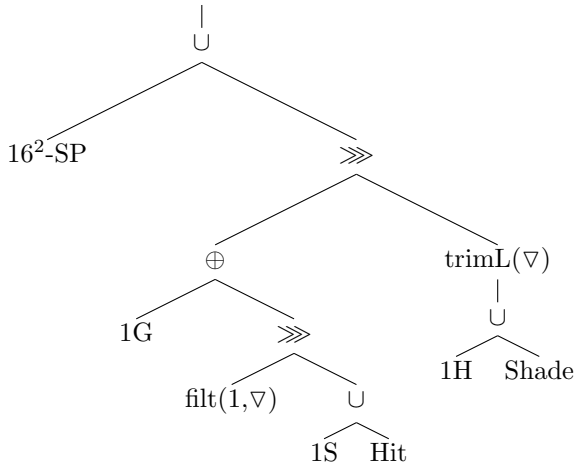
2 Example Trees

2.1 Tiled Forward

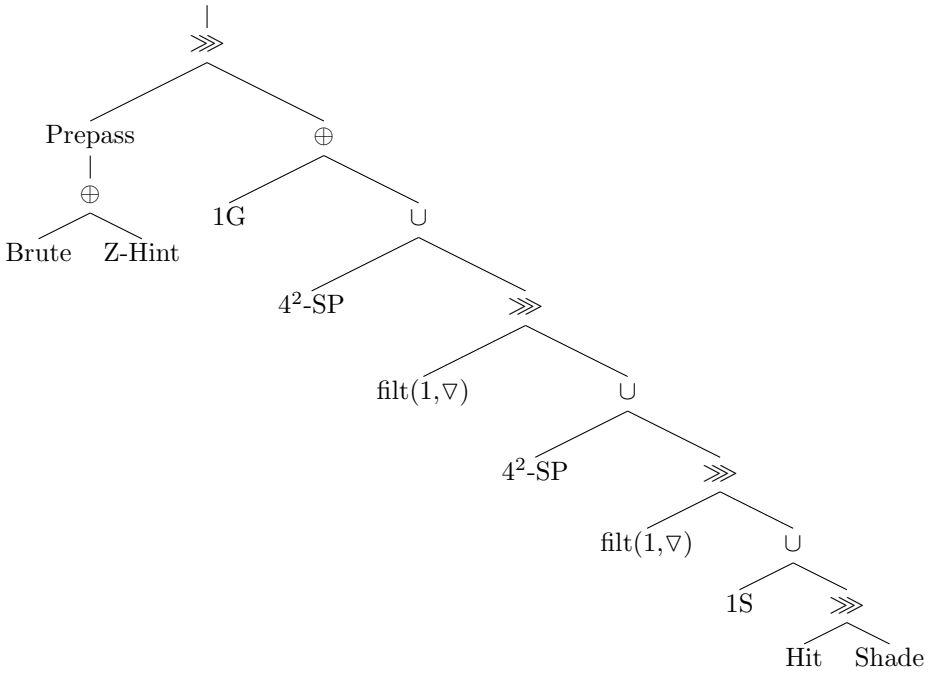
Here is the basic tiled rasterizer.



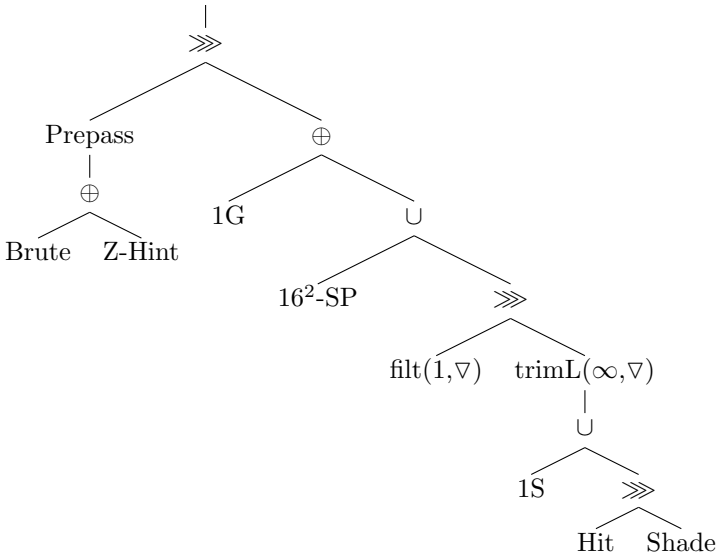
2.2 Tiled Defered Shading



2.3 Prepass With Double-Bounded Z Pyramid



2.4 Tiled Forward+ Shading



2.5 Quad-Tree over lights