Summary of High-Quality Surface Splatting on Today's GPUs

Dominik Schörkhuberl e1027470, Stefan Zaufl e0925357

March 19, 2015

1 Overview

Splat rendering is used to render the surfaces of point cloud data. In the paper 'High-Quality Surface Splatting on Today's GPUs' by Mario Botsch, Alexander Hornung, Mathias Zwicker and Leif Kobbelt a fast hardware implementation of this rendering technique is presented. It uses deferred rendering and a fast approximation of the EWA-prefilter to avoid aliasing artefacts.

2 Rasterization

For each point a splat in form of an ellipse is generated. Since ellipses can't directly be rendered with OpenGL, each point is expanded to a square region. The fragment-shader then simply discards fragments outside the ellipse. Additionally for each fragment in the ellipse a Gaussian weighting factor is computed, which is later used to blend between different splats.

3 Deferred Rendering

The rendering consists of 3 passes: a visibility pass, an attribute pass and a shading pass. In the first pass a depth map is generated that describes the surface of the point cloud data. Here no surface attributes are written, this pass focuses on detecting all visible points. The second pass rasterizes all splats again and computes weighing factors for the surface attribute such as color and normals which are stored in individual textures. Finally in the shading pass a fullscreen-rectangle is drawn and for each fragment we compute all attributes by dividing the accumulated values by the sum of

weights. The fragment-based shading can be applied for each light in the scene in this pass.

4 Fast EWA Approximation

To counter aliasing artefacts an approximation of the EWA-filter is applied. It reduces the amount of splats drawn by dropping all splats that are smaller than a certain radius(they used 2x2 pixels). Because the computation of the correct EWA-filter is very involving they used a union of the reconstruction and a simple low-pass filter.

5 Implementation

We plan to implement the deferred shading pipeline with simple Phong shading. Additionally we will also apply the EWA filter approximation. Regarding the technologies we decided on using C++ with openGL and related libraries.

Our program will first load a data-set containing 3D points, normals, splat axes and colors. For the described rendering passes we generate a framebuffer with several attached textures. Namely a depth-component and 2 float-color RGBA attachments: one for color and the weights, one for the normals. For the splat rasterization needed for pass 1 and 2 we start in the vertex shader. There we transform the points and the tangents into scree-space and estimate the size of the square splat patch. In the geometry stage we expand the point to a square. In the fragment shader all points were tested if they lay inside of the ellipse, if not these points get discarded. In the second pass a per point depth test is performed and the surface attributes get accumulated by additive blending. The third pass draws a fullscreen rectangle and accesses the values computed in the second pass and normalizes them by the accumulated weights. These values are then used for the final shading.