Rendering Leaves Using Hardware Tessellation

Jo Skjermo
Department of Computer and Information Science, NTNU

Abstract

This paper presents early work on a new approach for level of detail (LOD) rendering of leaves. The approach takes advantage of the new tessellation engine in computer graphics hardware to produce quads for leafs or leaf cluster billboards as needed in a single pass. The approach is based on area preservation and as such is conceptually simple. It runs fully on the GPU, freeing the CPU for other operations and also limits the number of draw-calls as one only has to draw at most a few patches per tree. The nature of the approach also makes it a prime candidate for further optimization by instancing, reducing the number of draw-calls even further.

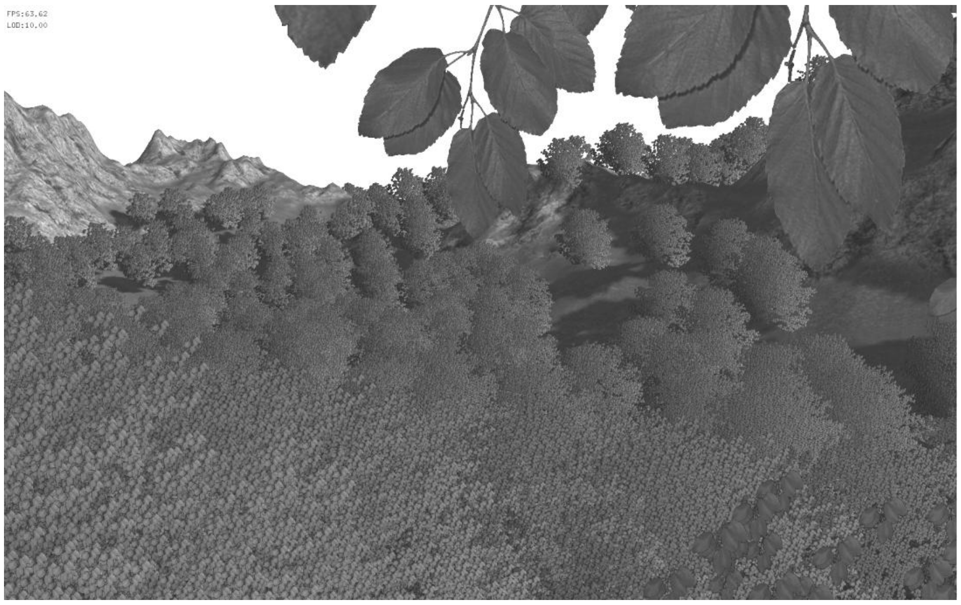

Figure 1 Frame from a scene with leaves on more than 1100 trees, drawn using one patch per tree, producing up to 4225 leaf quads per input patch.

Introduction

A fully grown tree can typically consist of several hundred of thousands of leafs. This means that in real-time computer graphics it's virtually impossible to render even a moderate amount of trees, without some way to reduce the number of polygons needed. One approach often used to reduce the number of polygons is to cluster leafs into billboards, as seen in [1], or impostors. Such approaches typically scales from using a single polygon with a billboard for the whole tree, including the trees branch structure, down to using hundreds or thousands of polygons per tree, where each has a billboard of only a few leaves.

However, even when clustering leaves into billboards, one can easily stall the graphics pipeline with too many polygons, especially in forested area. As the number of polygons for leaf billboards increases, so does the need for approaches that gracefully handles level of detail for leaf rendering.

Approach

The main concept used for the proposed approach is to maintain the overall area the leaves of a tree cover, while the number of leaves and the individual area each leaf covers are scaled based on distance to the observer. This is somewhat similar to the approach used in early versions of the commercial SpeedTree middleware for tree rendering, as described in [1]. However, where SpeedTree used the GPU to blend between fixed numbers of detail levels made during a preprocessing step, the proposed approach handles all scaling adaptively in real-time to minimize popping artifacts. To produce the required number of leaf quads per tree, and scale these, the new tessellation hardware shader steps in DX11 and OpenGL 4.0 supported GPU's was used. For the test implementation OpenGL 4.0 and GLSL with the GL_ARB_tessellation_shader [2] extension was used, on a GeForce GTX 480 graphics card.

The initial idea was to give a patch (a new draw primitive) as input to the tessellation steps, tessellate this into quads based on camera distance while maintaining the overall area, and then position each produced quad using a geometry shader based on positions stored as a texture buffer.

However, there is no way to easily generate a unique and stable index for each tessellated output polygon on today's hardware when using adaptive tessellation, as the tessellation pattern is vendor specific. A workaround for this problem is to use non-adaptive tessellation, and output points instead of quads to the geometry shader. In the geometry shader one can then use the produced u and v values together with the tessellation factor to calculate an index. This index is used to read in a leaf position, and generate a quad that is scaled based on the distance to the camera.

Implementation Details

As today's hardware can tessellate an input patch into 65x65 points (64x64 quads), a single patch can be used to feed 4225 points into the geometry shader. For each tree leaf quad positions are stored in batches of 4225 positions. The leaf positions for each tree are *randomized*, and then stored in a single texture buffer object on a per patch basis.

The overall approach is shown in Figure 2. For each tree, a single patch is drawn at the center position of the tree. This patch consists of 4 points making a rectangle, and the area of the patch covers the same area as all the leaves in the tree (included transparent areas in the leaf texture). If more than max 4229 leaf quads are to be produced one patch per 4229 quads must be drawn (with adjusted/reduced area per patch).

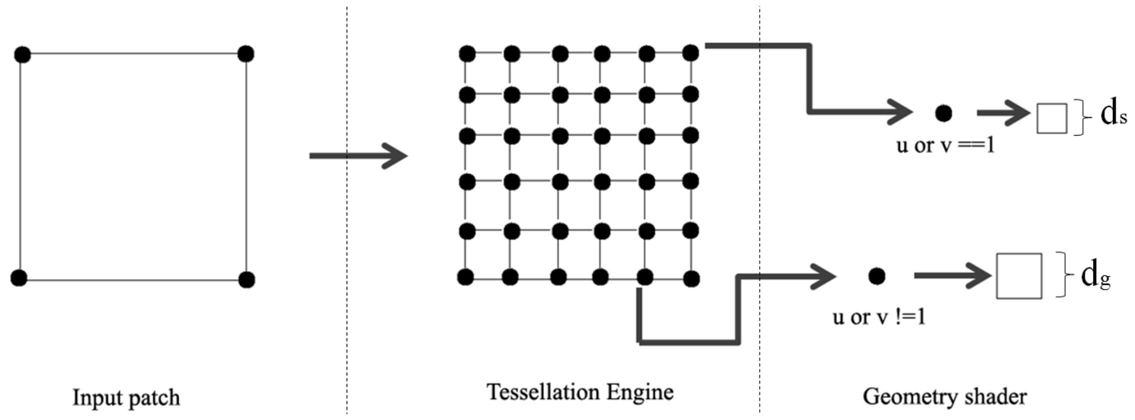

Figure 2 Patch input is tessellated into a number of points based on LOD level. The points are sent to the geometry shader, where they are used to produce quads with positions read from texture buffer memory and size calculated to maintain the area.

In the tessellation control shader a continuous LOD value is calculated based on the distance between the camera and the center of a patch. This LOD value is used for all tessellation factors for the tessellation engine for the patch. For the initial test application a LOD value is found using the linear interpolation between a minimum and a maximum tessellation factor controlled by a distance value (using the `mix` GLSL function defined as `mix(x,y,a)=x*(1.0-a)+y*a`). The distance value is a Hermite interpolation between a minimum and a maximum distance, based on the actual distance between the patch center and the camera using the `smoothstep` GLSL function (that performs a smooth Hermite interpolation with result clamped to be between 0 and 1).

In the tessellation evaluation shader one would normally do a parametric evaluation of each produced point, using the u and v values generated by the tessellation engine. Instead the output from this stage is made to be a single vertex together with the u and v value.

Each invocation of the geometry shader receives such a point as input. For each point a rectangular quad is generated, using an edge length calculated as shown in Equation 1. If the u and v values for the input are less than 1.0, the area is scaled down using d_s as the tessellation factor (and the number of quads) increases. However, if u or v is equal to 1.0, the size of the quad grows with d_g from 0, so that these quads appear to grow into the scene. Also, an alpha value is calculated to blend in the quad even more gently.

$$d_s = \frac{patchLength}{tessFactor}$$

$$d_g = \frac{patchLength}{tessFactor} * \quad -mix(1{,}0, ceil(tessFactor) - tessFactor))$$

Equation 1 Length of the sides. $patchLength$ is the length of the sides of the drawn patch, $mix(a, b, t)$ is the linear interpolation between a and b, given t

After the quad has been produced and scaled, it is positioned and rotated toward the camera. The position has to be read from the texture buffer, and the index used to look up a position is simply calculated as seen in Equation 2. We use gl_PrimitiveIDIn to find the location in the texture buffer for the leaves of the present input patch.

$$u_r = ceil(tessFactor * u)$$
$$v_r = ceil(tessFactor * v)$$
$$i = (\text{gl\_PrimitiveIDIn} * 65 * 65) + (u_r * 65 + v_r)$$

Equation 2 Calculating the index for each leaf quad position

Results and Future Work

Although the development of the proposed method is in a very early stage, a small test application has been made. A frame from this application can be seen in Figure 1. A single tree rendered with the approach can be seen in Figure 3. This application uses models from the XFrog software [3]. Initial testing indicate that the approach can maintain 60fps while producing 15-16 million vertices, but more testing are still needed at this time to evaluate the approach in a setting that is closer to an actual production system. It is however clear that on today's hardware the approach is fully limited by the performance of the hardware tessellation engine. Also, as the same tessellation engine will also have to handle all other models that need tessellation, the amount of patches one can use for leaf rendering is limited. However, we predict a rapid increase in the performance of tessellation on GPUs in the years ahead.

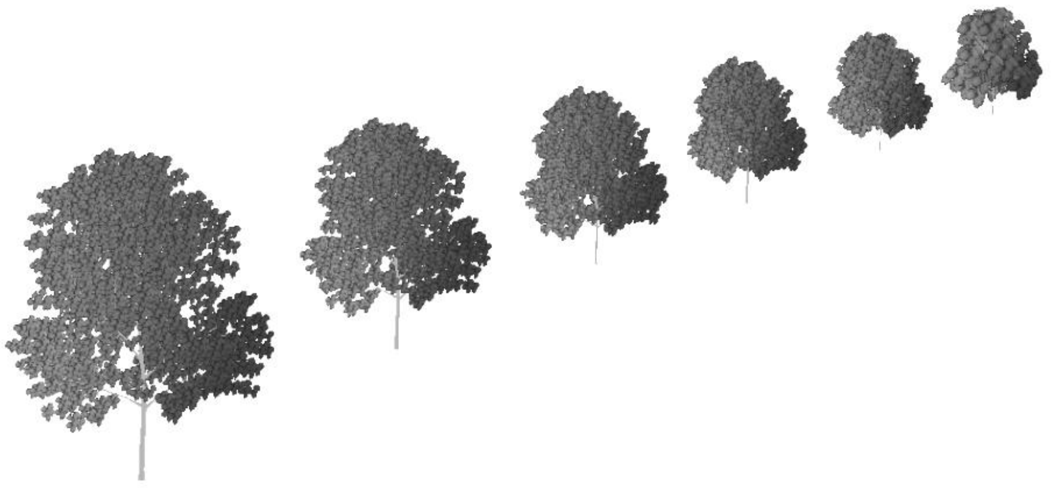
Figure 3 Several different level of details of a young Sweet Birch with branches rendered using the proposed approach, using one patch, from left: 3600, 2500, 1600, 400 and 100 quads produced.

A next step is to test the approach on trees models with full resolution, using a one-to-one mapping (one quad per leaf) between both position and orientation of the leaves and the produced quads. In addition, the approach should be expanded to handle fronds (small branch clusters in the tree handled as billboards).

Today, the test application only renders the leaves for each tree, not the stem and branches. If the models are made to have enough billboard position suitable for testing the proposed approach, the stems produced by the XFrog software are of a very high resolution and quite costly to render without simplification. In [4] it was suggested to use a single extremely simplified watertight mesh model for the stem and branches of a tree, and then tessellate this in hardware using Catmull-Clark Approximation or similar approaches, to increase the models resolution in real-time based on distance to the observer. Tessellation, together with displacement mapping for the bark and other features, can then produce models of very high resolution. A next step will be to merge these two approaches to hopefully be able to render trees in near full resolution with excellent mesh resolution even when observed at a human or toward insect scale, while maintaining a high frame-rate even when the trees are close to the observer.

References

[1] “Toward photorealism in virtual botany”, Whatley, D. (2005). In GPU Gems 2, ISBN 0-321-33559-7, pages 7–25. Addison-Wesley Professional.

[2] “GL_ARB_tessellation_shader”. OpenGL extension.
http://www.opengl.org/registry/specs/ARB/tessellation_shader.txt (valid 17. june 2010)

[3] “Greenworks xfrog, tree modelling software”. Commersial product,
http://www.xfrogdownloads.com.

[4] “Generation, Rendering and Animation of Polygon Tree Models”, Skjermo, Jo; Downing, Keith; Hallgren, Torbjørn; Sæther, Bjørn. NTNU-Trykk 2009 (ISBN 978-82-471-1701-9) 114 s. NTNU