To appear in 2000 SIGGRAPH/Eurographics Hardware Workshop

Adaptive View Dependent Tessellation of Displacement Maps

Michael Doggett, Johannes Hirche*

WSI/GRIS, University of Tübingen, Germany

Abstract

Displacement Mapping is an effective technique for encoding the high levels of detail found in today's triangle based surface models. Extending the hardware rendering pipeline to be capable of handling displacement maps as geometric primitives, will allow highly detailed models to be constructed without requiring large numbers of triangles to be passed from the CPU to the graphics pipeline. We present a new approach based on recursive tessellation that adapts to the surface complexity described by the displacement map. We also ensure that the resolution of the displaced mesh is tessellated with respect to the current view point. Our tessellation scheme performs all tests only on triangle edges to avoid generating cracks on the displaced surface. The main decision for vertex insertion is based on two comparisons involving the average height surrounding the vertices and the normals at the vertices. Individually, the tests will fail to tessellate a mesh satisfactorily, but their combination achieves good results.

We propose several additions to the typical hardware rendering pipeline in order to achieve displacement map rendering in hardware. The mesh tessellation is placed within the rendering pipeline so that we can take advantage of the pre-existing vertex transformation units to perform the setup calculations for our view dependent test. Our method adds only simple arithmetic and comparison operations to the graphics pipeline and makes use of existing units for calculations wherever possible.

CR Categories: I.3.1 [Computer Graphics]: Hardware Architecture, Graphics Processors; I.3.3 [Computer Graphics]: Picture/Image Generation, Display Algorithms

Keywords: Displacement Mapping, Graphics Hardware.

1 Introduction

Three dimensional graphics rendering hardware capable of rendering scenes with a large number of polygons is now widely available. Most systems support texture mapping, which adds surface detail via a two dimensional colour map that changes the colour at each pixel on a flat polygonal surface. To make the flat surface look as though there are small changes in the surface geometry, Blinn introduced Bump Mapping [1], which is becoming a standard feature in modern 3D graphics hardware.

* e-mail : {miked,jhirche}@gris.uni-tuebingen.de

Bump mapping does not really alter the underlying geometry, but only perturbs the normals on the surface. For small irregularities on a surface, bump mapping is ideal, since it does not increase the amount of geometry, but gives the impression of real textured surfaces. The limitation of bump mapping becomes obvious when the surface is parallel to the viewer and the Bump does not create a silhouette. Also as a surface moves in perspective space the shape created in the viewers mind by the bump map will not occlude other objects.

To add real geometric detail to a flat surface, displacement mapping, first introduced by Cook [2], can be used. As an example Figure 1(a) shows a flat plane with a colour texture applied, which is displaced using the displacement map in Figure 1(b), where white represents a high displacement and black represents no displacement. The application of the displacement map to the plane results in the the displaced surface shown in Figure 1(c). An advantage of using displacement maps is that highly detailed triangle surface models, such as those generated by 3D scanning technologies, can be modelled using only a simple base surface and displacement maps. A compact surface representation using scalar-valued displacements over a smooth domain surface is presented by Lee [11]. Displacement mapping has been used frequently to add geometric surface detail to objects [17, 12, 9], and is commonly found in commercially available software renders. But unlike bump mapping, displacement mapping has not been implemented in hardware, due to the high computational cost.

(a) (b) (c)

Figure 1: A plane displaced with a text displacement map.

.

Adding displacement map rendering to currently available hardware architectures presents several problems. To apply a displacement map to a triangle mesh involves re-triangulating the original mesh and displacing the vertices accordingly. If the base triangle mesh has a coarser resolution than the displacement map we need to re-tessellate the mesh according to the surface defined by the displacement map. This re-tessellation can be performed while rasterizing the triangle as presented by Doggett and Kugler in [4] and Gumhold [7]. The problem with these approaches is that either a large number of triangles is generated [4] or new coordinate systems and complex calculations in a non-standard rasterizer are required to control the pipeline [7].

We propose to tessellate the individual triangles sequentially by recursively adding vertices along edges in order to achieve an adaptively tessellated surface. We check local surface variation between vertices by comparing normals. At the same time we check the av-

erage displacement over an area around the vertices to avoid the aliasing caused by sampling at vertices by using Summed-Area Tables as presented by Crow [3]. These two tests applied individually do not generate good results, since the normal test results in aliasing and the averaged height test misses small details, but the combination of the two tests produces good results. To give the user control over the size of triangles generated with respect to the current view point, we test the screen size of triangles against a user threshold. This ensures that recursive tessellation will stop when the user does not require further detail. A displacement map models a discrete surface which has a limit of resolution. We also check if this limit is reached before generating more triangles, to ensure no more triangles are generated then there is detail in the original displacement map.

To minimise the additional hardware required to render displacement maps, we exploit existing pipeline units wherever possible. A bump mapping unit is used for displaced surface normal calculation and extra units are carefully placed within the pipeline to make use of the operations performed by the transformation unit. The remaining computations only require simple operations such as addition, division by two and comparisons.

1.1 Previous Work

1.1.1 Bump Mapping

Bump mapping, introduced by Blinn [1], perturbs surface normals on a surface as if a height field displaced the surface in the direction of the original surface normal. Since bump mapping only changes the appearance of an object, it makes certain approximations. Standard techniques for bump mapping assume that the bumpiness is only microdisplacements and hence assume the magnitude of the height field is negligible.

Several hardware based approaches to implementing bump mapping have been presented in recent years [5, 13, 10]. Peercy presents an implementation for high-end 3D graphics hardware [13] that supports bump mapping within the context of per-fragment lighting operations. Kilgard presents an approach to bump mapping that takes advantage of currently available low-cost graphics hardware in [8]. Although our method requires the use of a bump mapping unit, we will not go into the implementation details in this paper.

1.2 Terrain Modelling

Terrain modelling using height fields is closely related to the rendering of displacement maps. Many algorithms are available for approximating terrains and height fields using polygonal meshes. These algorithms approximate the height field with a mesh of triangles, also known as a triangulated irregular network, or TIN. Garland [6] analyses several of these algorithms including the greedy insertion algorithm. While very good results are achieved these techniques relie on global mesh information making hardware implementation expensive.

1.3 Displacement Mapping

Displacement mapping [2] perturbs a parametric surface along its normals based on a height field to create a new surface. The base surface is defined by a bivariate vector function $\mathbf{P}(u, v)$ that defines 3D points (x, y, z) on the surface. The displacements from the surface are defined by a bivariate scalar function $D(u, v)$ and the normals on the base surface $\mathbf{P}(u, v)$ by $\hat{\mathbf{N}}(u, v)$. The points on the new displaced surface $\mathbf{P}'(u, v)$ are defined as follows

$$\mathbf{P}'(u, v) = \mathbf{P}(u, v) + D(u, v)\hat{\mathbf{N}}(u, v) \quad (1)$$

where $\hat{\mathbf{N}}(u, v) = \frac{\mathbf{N}(u,v)}{|\mathbf{N}(u,v)|}$.

A cross section of an example displacement mapped surface is shown in Figure 2, where $\mathbf{N}'(u, v)$ is the normal to the displaced surface.

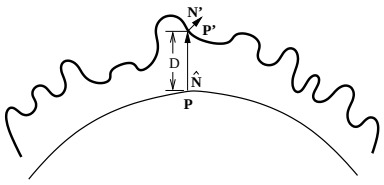


Figure 2: A cross section of a displaced surface.

In Section 2, we describe the calculations required for our adaptive view dependent tessellation algorithm and in Section 3, we explain how these operations are implemented in a graphics pipeline. Section 4 presents the results and Section 5 concludes and presents future work.

2 Adaptive View Dependent Tessellation

Our algorithm recursively tessellates the triangles given in the base mesh or surface $\mathbf{P}(u, v)$ by inserting vertices along edges of triangles. A base triangle that has been recursively adaptively tessellated to increase the number of vertices in the area of height change is shown in Figure 3.

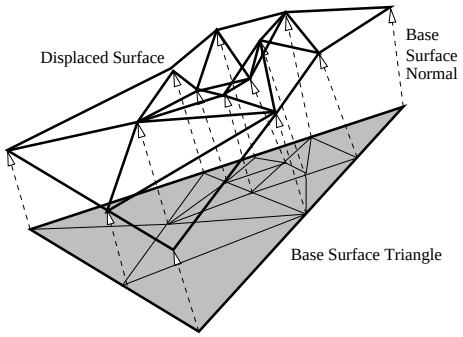


Figure 3: A triangle from the base surface is recursively tessellated to create a displaced surface.

2.1 Tessellation based only on edge information

Displacement mapping requires a coarse triangle mesh that approximates the surface to be modelled with a displacement map containing the finer geometric detail. A re-meshing step is required to generate a finer mesh that better represents the desired surface. While many re-meshing algorithms exist [6], few are suitable for hardware implementation. For example, if tessellation decisions are based on information local to one triangle then the neighbouring triangles would need access to this information requiring processing of each triangle to have access to the entire triangle list. Random access to memory of this nature is expensive when implemented in hardware. To avoid such memory requirements, we limit our vertex insertion decision to using only the information available within the current triangle. Furthermore if vertex insertion decisions use all three vertices of a triangle then the common edge between two adjacent triangles may have a different insertion decision due to the

third triangle vertex. Vertices inserted along the edge of one triangle but not inserted on the same edge of a neighbouring triangle are known as t-vertices. To avoid cracking in the displaced surface t-vertices must be avoided. To ensure no t-vertices are generated we limit our vertex insertion decision to information contained in only the two vertices that subtend the edge in question. The tradeoff is that calculations for each edge are performed twice.

Re-meshing is performed by inserting vertices at the midpoint of edges if the midpoint meets a series of conditions. First the new vertex $\mathbf{P}_{12}$ between the two vertices $\mathbf{P}_1, \mathbf{P}_2$ of an edge is calculated by averaging the two vertices. The texture coordinates $\mathbf{U}_{12}$ and the mesh normal $\mathbf{N}_{12}$ are similarly calculated. The normal on the displaced surface, $\mathbf{N}'$ can be calculated from the original mesh normal, $\mathbf{N}$ and the normal calculated from the displacement map, $\mathbf{N}_D$, using the bump mapping operations described by Schilling [15]. The normal to the displacement map, $D(u, v)$ can be precomputed using finite differencing and stored in a normal map in a similar fashion to RGB values stored in texture maps. The displaced surfaces and normals are shown in Figure 2.

2.2 Surface Normal Variance Test

Using the new surface normal, $\mathbf{N'}_{12}$, we compare it's components with those of the two displaced vertex normals, $\mathbf{N'}_1$ and $\mathbf{N'}_2$. If the difference between any of the components is greater than a set threshold, then a vertex is added at $\mathbf{P}_{12}$ (*Normal Test*). The boolean value for the Normal Test, nt is defined as

$$\begin{aligned} nt \quad = \quad & (\mathbf{N'}_{12x} - \mathbf{N'}_{1x} < nthr) | (\mathbf{N'}_{12y} - \mathbf{N'}_{1y} < nthr) | \\ & (\mathbf{N'}_{12z} - \mathbf{N'}_{1z} < nthr) | (\mathbf{N'}_{12x} - \mathbf{N'}_{2x} < nthr) | \\ & (\mathbf{N'}_{12y} - \mathbf{N'}_{2y} < nthr) | (\mathbf{N'}_{12z} - \mathbf{N'}_{2z} < nthr) \end{aligned}$$

where $nthr$ is the normal threshold and the $|$ symbol represents the logical *or* operation.

The Normal Test is subject to aliasing because it uses point sampling and can easily miss changes in height. A simple example of this is shown in Figure 5(a). The advantage of the Normal Test is that it is extremely sensitive to small scale changes in the surface. The user is required to predetermine the threshold value $nthr$ and supply this value with the mesh to achieve the required level of detail.

2.3 Local Area Average Height Test

To detect average changes in the height of the displacement map, a second test is performed that compares the displacement over an area using Summed-Area Tables, introduced by Crow [3]. A Summed-Area Table is a two dimensional array containing at each cell the sum of all values that fall inside the rectangle formed by that cell and one corner of the array. To calculate the sum of all values within a rectangular area in the table only the four values at the corners of the area are needed. The Summed-Area Table can be represented as a bivariate function $SAT(x, y)$ that returns the sum of all heights within the region $(0 \rightarrow x, 0 \rightarrow y)$, where the origin is in the bottom left of the table. The sum of the values within a rectangular area are calculated using the function $S(\mathbf{Z})$ defined as

$$\begin{aligned} S(\mathbf{Z}) \quad = \quad & SAT(x_{tr}, y_{tr}) - SAT(x_{tl}, y_{tl}) - \\ & SAT(x_{br}, y_{br}) + SAT(x_{bl}, y_{bl}) \end{aligned} \qquad (2)$$

where $\mathbf{Z}$ is $(x_{tr}, y_{tr}, x_{tl}, y_{tl}, x_{br}, y_{br}, x_{bl}, y_{bl})$, the corners of the rectangular area in the Summed-Area Table, using the subscripts tr top right, tl top left, br bottom right, bl bottom left for the four corners of the rectangle .

A Summed-Area Table can be precalculated for the displacement map and the sum of all heights over an area at vertices $\mathbf{P}_1, \mathbf{P}_2$ and $\mathbf{P}_{12}$ can be calculated. To calculate the four corner points of the rectangle around the vertices of the edge we use the texture coordinates at the vertices $\mathbf{U}_1, \mathbf{U}_2$ and $\mathbf{U}_{12}$ as shown in Figure 4. Using

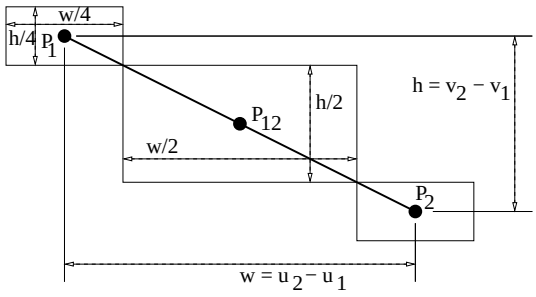


Figure 4: Using the texture coordinates of the vertices to calculate the summed height.

the texture coordinates we can calculate the difference of the areas and compare them with a threshold. This test is called the *Summed Height Test* and its boolean value, sht, is defined as

$$sht = \left(\frac{S(P_{12})}{2} - (S(P_1) + S(P_2)) \right) < shthr$$

where $shthr$ is the summed height threshold. This test misses some cases that the Normal Test detects as shown in Figure 5(b). There are cases that cannot be detected by both tests. If the displacement map is highly regular, e.g. high frequency such as a sinusoidal function, then the Normal Test could always sample identical normals and the height averaging would cancel itself out. Typically a displacement map and associated triangle mesh are created together by the author with the purpose of modelling a particular surface so these cases can be avoided. The combination of the Normal Test and Summed Height Test with appropriate thresholds is capable of re-meshing the small detail in a surface without missing height changes in smooth surfaces resulting in effective filtering of the displacement map.

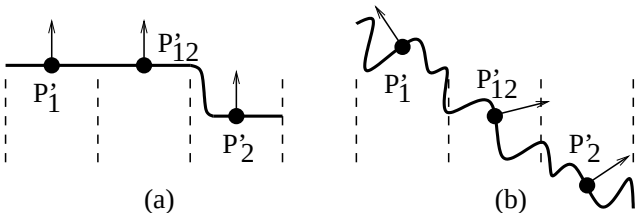


Figure 5: The solid line represents a contour line across the displacement map between the texture coordinates of the vertices of one edge of a triangle with the newly inserted point in the middle. The dashed lines indicate the area over which the height is averaged to calculate the Summed Height value. In (a) the Normal Test fails, but the Summed Height Test succeeds. In (b) the Normal Test succeeds but the Summed Height Test fails.

2.3.1 Displacement Map Filtering

The standard approach for texture filtering is mipmapping as presented by Williams [16]. This provides an effective means of retrieving levels of detail in colour textures that match the screen size of an object. But for displacement mapping, the averaging effect

of mipmapping will smooth over areas of detail in the displacement map. In [7], they propose to use mipmapping, but with a maximal filter to overcome this. The Summed Height Test can still miss details due to averaging over the height, but the normal test will find these details. And conversely, when the Normal test fails to detect height change over a smooth surface, the Summed Height Test will detect the change. This combination results in an effective filtering of the displacement map.

2.4 View Dependent Resampling

To achieve a view dependent re-sampling that ensures no new triangles are generated once a certain screen size has been reached, we perform a test based on the size of the current edge in Screen Space. This test can be performed after the vertices $\mathbf{P'}_1$ and $\mathbf{P'}_2$ of the current edge are transformed into Screen Space by carefully placing the re-meshing hardware within an existing graphics pipeline. Calculating the Euclidean distance requires expensive square and square root operations, so we only use a Manhattan distance calculation between the two screen space vertices. This Manhattan distance is compared with a threshold measured in number of pixels. This test is called the *View Test* and if the Manhattan distance is greater then a threshold measured in number of pixels, $vthr$ then the boolean value vt is set to true.

2.5 Refinement Limit Test

Height fields can be rendered by first generating a mesh by inserting the points defined by the height field into a mesh using algorithms such as the Greedy Insertion Algorithm [6]. These algorithms pass over the complete mesh and determine the point of maximum error and insert that point. In the algorithm presented here, we only insert points calculated from the vertices of triangle edges as they are easy to compute and only depend on the local information of a single edge. If we want to insert points inside a triangle, then the other vertex in a triangle is required to calculate the interpolated normal at the point inside the triangle. Another problem occrs when an adjacent triangle calculates that a point to be inserted is outside its triangle and is actually inside an adjacent triangle, then the triangle needs access to the vertex in the adjacent triangle.

The alternative of only inserting points on edges leads to the problem that the points defined by the displacement map are only approximated and several points might be inserted close to the original point due to this approximation error. To stop the recursive algorithm from over inserting points when the original resolution of the displacement map is reached, we perform a test on the texture coordinates of the new point. We compare the integer values of the texture coordinates after they are scaled to the resolution of the displacement map. If the value of both integer parts of the texture coordinates at the newly inserted vertex $\mathbf{U}_{12}$ are equal to either of the scaled values of the two original vertices $\mathbf{U}_1$ and $\mathbf{U}_2$, then the new vertex is not inserted and the boolean value, ct is set to false. The test is called the *Tex Coord Test*. This test stops recursive subdivision, once the resolution of the original displacement map is reached. We have found that models with areas of high variance of the displacement map, such as the hair on the model of Volker Blanz's head (see Figure 9), can reduce their final triangle counts by up to 50% by incorporating this simple test.

An alternative to performing the Tex Coord Test is to precalculate the points on the new surface $\mathbf{P'}(u, v)$ and store them with the displacement map. These values can then be inserted instead of inserting midpoints resulting in a more accurate representation of the original displacement map. The problem with this technique is that the displacement map cannot be interactively scaled because the new surface has already been pre-calculated.

2.6 Tessellation

Once all the tests have been performed for an edge of a triangle the decision to split the edge is calculated using the following logic equation

$$split = (nt|sht)\&vt\&ct$$

where | represents the logical *or* operation and & represents the logical *and* operation. This $split$ value is calculated for each edge and depending on the number of edges that have to be split an appropriate tessellation is chosen from Figure 6(a-c). Figure 6(d) shows two other possibilities for tessellating case 6(c), but we found that these two triangulations increased the occurance of long and thin triangles so we used the triangulation shown in (c) instead.

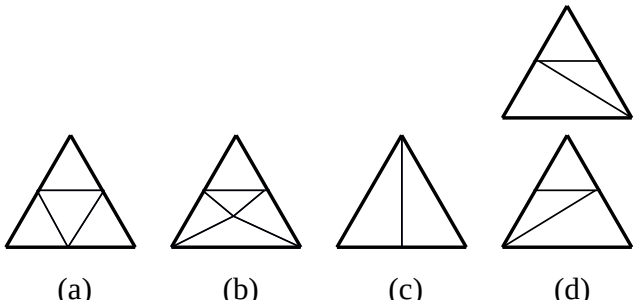


Figure 6: The three triangulations used for tessellation. (a) is used if three edges are to be split, (b) is used if two edges are split and (c) is used when one edge is split. (d) Two other possibilities for splitting two edges that can produce long thin triangles.

2.7 Curved Surface rendering using Displacement Maps

One problem for the rendering of displacement maps in hardware is that modellers are not used to thinking in terms of generating models from displacement maps. In order to handle higher order primitives such as quadric curves, bezier surfaces and NURBS using our algorithm, we have incorporated into our displacement mapping software simulation the capability to sample a quadric surface and render it using displacement mapping. The current software implementation has only limited capabilities, since it can only render NURBS which only elevate their parameter space.

The only information that can be stored in the NURBS surface is the height or displacement from a base mesh. Applied to a suitable base mesh, this already provides enormous flexibility for designing and modelling by exploiting the high triangle performance of available graphics cards. Although this sampling approach uses the host to compute the displacement map, it would be relatively straightforward to couple a hardware implementation of a curved surface evaluation – like that used in the OpenGL pipeline proposed by Rockwood [14] – with our hardware design, if they become available in the future. This would enable the rendering of curved surface models at high speed and without the need for new and expensive scanline converters.

3 Hardware Architecture

To realise hardware rendering of displacement maps, we suggest the introduction of several units into a standard rendering pipeline such as the OpenGL pipeline [18]. These new units and their positioning relative to the usual units in a pipeline are shown in Figure 7. The OpenGL *Clipping, Perspective, and Viewport Application*

stage is represented by our Transform stage. The OpenGL pipeline places Lighting inside the Vertices operations unit before transformation to screen space to achieve per-vertex lighting. Modern implementations of the graphics pipeline such as NVIDIA's GPUs place the lighting calculations after the transformation to improve per-fragment lighting capabilities. We have also moved the Lighting stage after the Transform stage, to allow our re-meshing and normal calculations to occur in world space.

We add four new units to the graphics pipeline including a *Get Triangle, Calculate New* $\mathbf{P}$*, Displace Vertex, Tessellation Tests* and *Tessellation* stage. Also storage for the displacement map and a triangle queue are added at the top level. The Get Triangle Unit is responsible for retrieving triangles from either the host CPU or the Triangle Queue. The Get Triangle Unit also checks which vertices and normals of incoming triangles have already been transformed into screen space and only sends the new vertices and their normals through the Transform unit.

The Calculate New $\mathbf{P}$ stage takes two vertices of a triangle and averages them to find the midpoint. This midpoint passes through the Displace Vertex stage which calculates the new surface point $\mathbf{P}'(\mathbf{u}, \mathbf{v})$ as defined in Equation 1. This requires reading the displacement *D* from the displacement map, multiplying it by the surface normal $\hat{\mathbf{N}}$ and adding the result to the current surface point $\mathbf{P}$. This operation is only performed for new vertices which are extracted from the current triangle information by the Get Triangle stage. Once the displacement is performed, the newly displaced vertex is passed through the Transform stage and transformed into screen space. The Tessellation stage takes the vertex information of the current triangle, the new vertices to be inserted, and constructs new triangles based on the triangulations shown in Section 2.6. These new triangles are inserted into the Triangle Queue, a FIFO queue, where they are read back into the pipeline by the Get Triangle unit. The Triangle Queue requires either on-chip memory or off-chip memory requiring an additional reading and writing unit. Either implementation is a significant requirement for this algorithm.

3.1 Tessellation Tests Architecture

The Tessellation Tests unit calculates new texture coordinates and normals and the four edge tests. The unit is pipelined and each new vertex, $\mathbf{P}_{12}$, $\mathbf{P}_{23}$ and $\mathbf{P}_{31}$, is sent through the pipeline sequentially, requiring only one pipeline. If faster performance was required due to multiple Transform pipelines then the unit could be replicated three times. The architecture of the Tessellation Tests unit is shown in Figure 8 with inputs and outputs labelled for testing the edge between vertices $\mathbf{P}_1$ and $\mathbf{P}_2$.

3.1.1 View Test

The simplest test in the Tessellation Tests is the View Test. This test is done in screen space by calculating the Manhattan distance for the current edge using the displaced transformed vertices. For triangles that are being re-tessellated, the transformed vertex values will have already been calculated in the previous pass, so this data will be immediately available. But for new triangles, this stage will have to wait until the values are available after the Transform stage. This waiting time can be easily accommodated, since for each new triangle the tests must wait until not only the original vertices are transformed, but also the newly inserted vertices are transformed. Also the ratio of new triangles to re-tessellated triangles is quite low so this wait time will happen infrequently. The result of the View Test, vt is passed to the Pass or Tessellate unit.

3.1.2 Tex Coord Test

The calculation of the texture coordinates for the new vertices is required by several subsequent units. The *Calculate New U* calculates the new texture coordinates using an add and division by two. The integer part of the new texture coordinate up to the resolution of the displacement map are compared with the integer part of the original texture coordinates of the edge's vertices. The test is calculated by the *Tex Coord Test* unit and the result passed onto the Pass or Tessellate unit.

3.1.3 Summed Height Test

The new texture coordinates are used by the *Height Test* to look up and calculate an average height from the *Summed Height Table*. First the difference between the texture coordinates at the vertices are calculated and divided by 4 and 8 using logical shift operations. The results are combined with the vertex texture coordinates to calculate the four values required to access the Summed Height Table. The calculation of the average height for each point using the Summed Height Table will take four cycles since each value must be read and combined using Equation 2. The values are then combined and divided according to the Summed Height Test and the difference compared with the *shthr* to determine if there is a significant change in height. The result is passed to the Pass or Tessellate unit.

3.1.4 Normal Test

The other operation of the Tessellation Tests unit is the Normal Test. This test requires first the calculation of the new normal $\mathbf{N}_{12}$ using an add and division by two. Then the new normal must be normalised before being bump mapped. Normalisation is an expensive operation, but implementations are available in most graphics pipelines. Then using the new texture coordinates, the normal to the displacement $\mathbf{N}_D$ is read from a precomputed normal map. The new surface normal, $\mathbf{N}'_{12}$ is calculated by perturbing the surface normal, $\mathbf{N}_{12}$, by the displacement map normal, $\mathbf{N}_D$, using a bump mapping hardware unit. As mentioned earlier several approaches to bump mapping have been proposed and could be used to provide the operation required here. After calculating the new displaced surface normal its components are subtracted from those of the normals from the displaced vertices $\mathbf{N}'_1$ and $\mathbf{N}'_2$. The difference is compared to $nthr$ resulting in the boolean value nt which is passed to the Pass or Tessellate unit.

3.1.5 Pass or Tessellate Unit

The Pass or Tessellate unit combines the results of the four tests to determine if the edge is to be split. If the edge is to be split then the new vertex $\mathbf{P}'_{12}$ is indicated for insertion into the current triangle to the Tessellation unit. For each vertex that was part of the original triangle, the Pass or Tessellate unit passes the following set of values to the Tessellation unit : $\mathbf{P}_w, \mathbf{P'}_s, \mathbf{N}, \mathbf{N}', \mathbf{N'}_s, \mathbf{U}$ where the subscript $_w$ means the vector is in world space and the subscript $_s$ means the vector is in screen space. For the newly inserted vertices, it has all of the above values except the $\mathbf{N'}_s$ value which is calculated on the triangles next pass through the pipeline. If the triangle has no new vertices to insert then only the following values are sent to the Lighting unit $\mathbf{P'}_s, \mathbf{N'}_s, \mathbf{U}$. The Pass or Tessellate unit can also perform backface culling on the new triangles since they have been generated with screen space normal values.

The most complex unit in the Tessellation Tests unit are those required for the Normal Test which include a bump mapping operation and a normalisation. Both units exist in most graphics pipelines and can be expected in most pipelines in the future. Also memory and a controller for the buffering in the Triangle Queue is required.

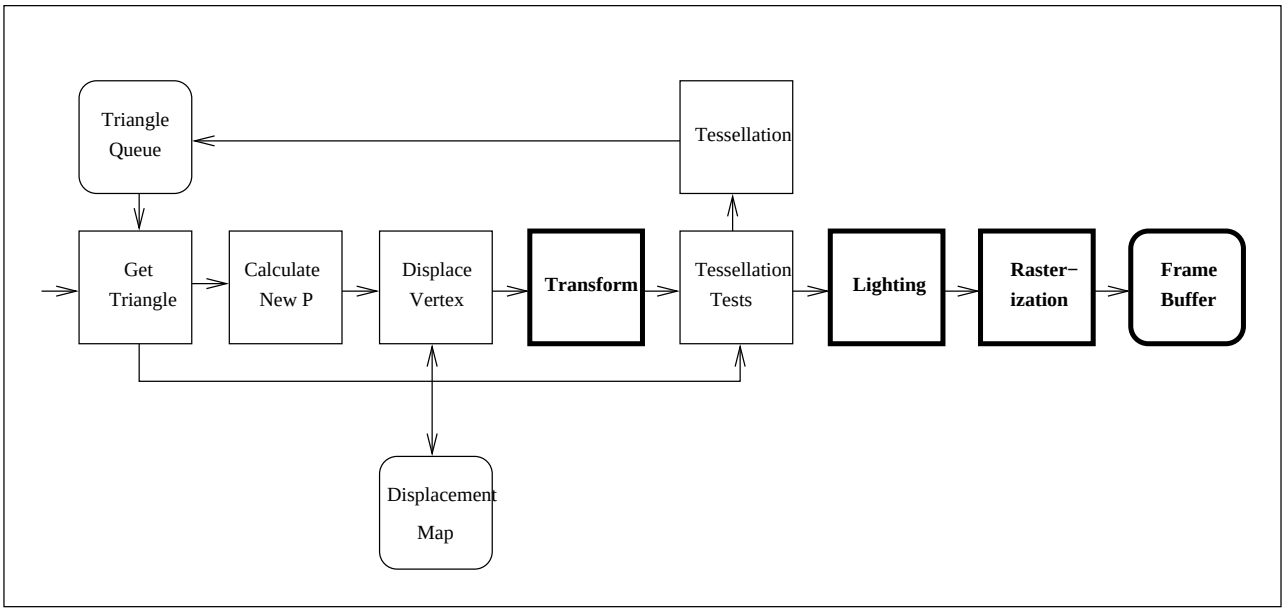


Figure 7: A graphics pipeline with new units added to perform displacement map rendering.

The other units only require addition, subtraction, divisions by powers of two, compare operations and other logic functions. The latency involved in the Tessellation Tests unit can be compensated by the latency of the Transform unit since both pipelines run in parallel. The major additional latency to the overall graphics pipeline is in the Displace Vertex unit which requires a multiply and accumulate.

4 Results

To demonstrate the effectiveness of our adaptive tessellation algorithm, we have implemented the architecture in Figure 7 and Figure 8 using a software simulation. To take advantage of OpenGL functionality, we use the feedback render mode to perform the transform calculations. We have rendered several models with our technique to demonstrate the range of datasets that it can handle and possibilities provided by displacement mapping.

To demonstrate the adaptive nature of the tessellation and the effectiveness of combining the Normal Test and Summed Height Test, a flat plane consisting of only 18 triangles was displaced with a half donut displacement as shown in Figure 10. In Figure 10(a), we can see the aliasing errors associated with the Normal Test, while in Figure 10(b), some edges have not been split because the averaging effect of the Summed Height Test has not detected the local change in surface orientation. The combination of the two tests is shown in Figure 10(c) where both the local surface orientation and average change in height is detected producing an adaptively tessellated displaced surface.

The view dependent capabilities of our algorithm are demonstrated using a cyberware scanner model of Volker Blanz's head. The scanner generates a texture and displacement map parametrised in cylindrical coordinates with a resolution of 512×456 texels. We use a cylinder for a base mesh consisting of 1800 triangles. The model rendered at three different distances from the view point is shown in Figure 9. In Figures 9(a,b,c), the image rendered shows the distance of the model from the view plane. In Figures 9(d,e,f), a wireframe close-up of the region around the nose, eyes and mouth using the respective meshes from (a,b,c), can be seen. In Figure 9(d) the high level of detail can be seen in the tessellation around the eye including the eye brow. Both the view dependent tessellation as well as the effective adaptive techniques are demonstrated. The adaptive nature of the algorithm can be seen in the level of detail around the nose, eyes and mouth where there is increased surface detail.

To evaluate the algorithm on terrain models, we used the Crater Lake (West half of Crater Lake, Oregon) Digital Elevation Map (DEM), which has dimensions of 366×459 from Garland [6]. In Figure 11(a) we can see that a small island inside the crater and the edge of the crater is represented with a high level of detail, while the water level in the crater is left at the resolution of the original mesh.

An example of the donut displacement map applied to the well-known Utah teapot is shown in Figure 11(b). We are also investigating the possibilities of saving the mesh state from one displacement map and adding another displacement to it as shown in Figure 11(c) where the base donut is a displacement map. While this works in our software simulation we are still considering the hardware implications of such an operation.

5 Conclusions and Future work

In this paper, we have proposed additional units for a standard graphics pipeline to enable the rendering of displacement maps in hardware. Our technique encompasses adaptive view dependent remeshing, driven by the displacement map's surface complexity and user defined thresholds. The additions to the hardware pipeline would only require standard features such as bump mapping and simple arithmetic units. We have tested our algorithm on several real and synthetic datasets and it is capable of generating low triangle count meshes, by adaptively subdividing triangles. Our tessellation scheme takes into account several factors including displacement, surface variation, view point and original mesh resolution. Backface culling cannot be applied to the original base mesh, because the displacement might generate triangles that are visible. Our tessellation results in normals in screen space, so backface culling can be applied after tessellation and before rasterization.

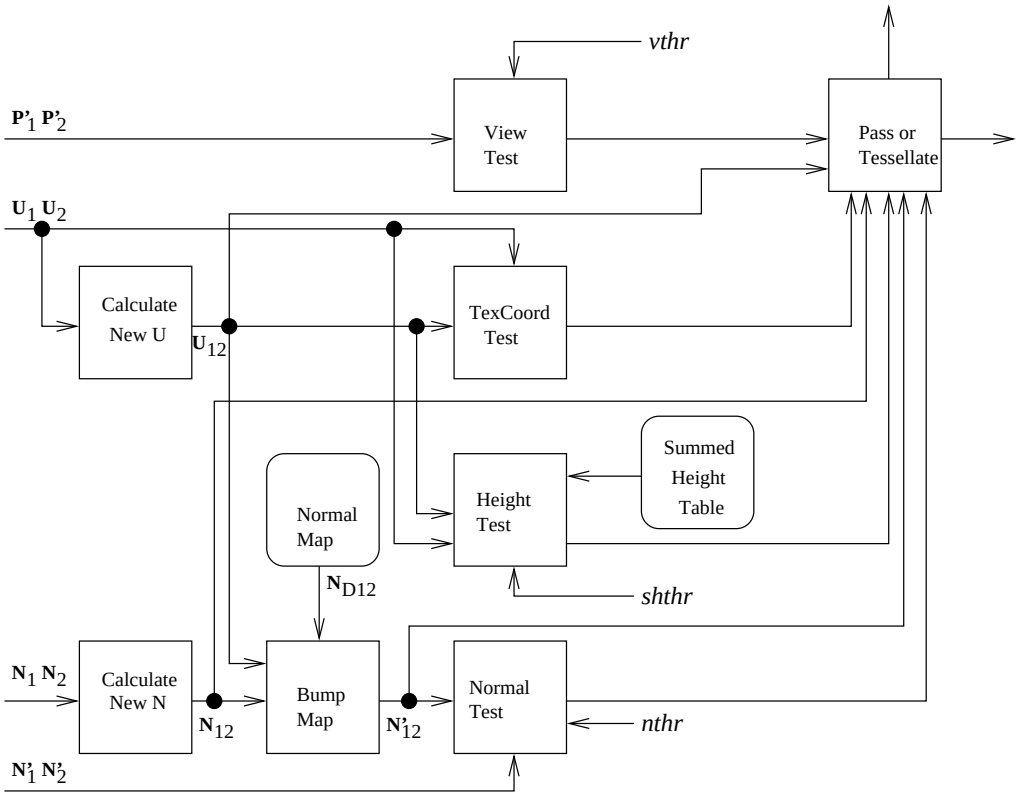


Figure 8: Architecture for the Tessellation Tests unit.

Our combination of the Normal Test with the Summed Height Test also provides an effective filter for the displacement map, which is not possible using maximal valued mipmapping. Using displacement mapping as a rendering primitive can significantly reduce the bottleneck of sending triangle data to the graphics engine. As displacement maps are used increasingly to model the geometric detail on surfaces, hardware support for rendering these displacement maps will become a sought-after feature in graphics accelerators.

5.1 Future work

There are several avenues of future work that are possible from what has been presented in this paper. Scaling the displacement map by introducing a scaling factor s into equation 1 is done as follows $\mathbf{P}'(u,v) = \mathbf{P}(u,v) + sD(u,v)\hat{\mathbf{N}}(u,v)$. The capability to scale the displacement map already exists within our software simulation, but requires that the displacement value used in the Displace Vertex unit is appropriately scaled and that the normal map is calculated using finite differences and is normalised for each new scale value of the displacement map. We are currently investigating the implementation details for this extension to our approach. Another interesting extension to our work includes tests on the surface normals to determine which triangles are located within or near the Phong highlight to improve the tessellation of these area when per-vertex lighting schemes are used.

Acknowledgements

This work has been funded by the SFB grant 382 of the German Research Council (DFG). Thanks to Anders Kugler, Michael Meißner, Dirk Bartz, Urs Kanus and Andreas Schilling for proof-reading and helpful discussions. Thanks to Volker Blanz for use of the digitized model of his head.

References

[1] Jim Blinn. *Jim Blinn's Corner: A Trip down the graphics pipeline*. Morgan Kaufmann, 1996. Chapter 17: Hyperbolic Interpolation.

[2] Robert L. Cook. Shade Trees. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):223–231, July 1984. Held in Minneapolis, Minnesota.

[3] Franklin C. Crow. Summed-Area Tables for Texture Mapping. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):207–212, July 1984. Held in Minneapolis, Minnesota.

[4] Michael Doggett and Anders Kugler. A Hardware Architecture for Displacement Mapping using Scan Conversion. Technical Report WSI–99–12, Wilhelm-Schickard-Institut für Informatik, University of Tübingen, Germany, 1999.

[5] I. Ernst, D. Jackél, H. Rüsseler, and O. Wittig. Hardware-supported bump mapping. *Computers and Graphics*, 20(4):515–521, 1996.

[6] Michael Garland and Paul S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[7] Stefan Gumhold and Tobias Hüttner. Multiresolution rendering with displacement mapping. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 55–66, August 1999.

[8] Mark J. Kilgard. A Practical and Robust Bump-Mapping Technique for Today's GPUs. Technical report, NVIDIA Corporation, www.nvidia.com, February 2000.

[9] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. *Proceedings of SIGGRAPH 96*, pages 313–324, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[10] Anders Kugler. IMEM: an intelligent memory for bump- and reflection-mapping. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 113–122, August 1998.

[11] Aaron Lee, Henry Moreton, and Hugues Hoppe. Displaced subdivision surfaces. In *Computer Graphics, Proc. of SIGGRAPH 2000*. ACM, 2000.

[12] Kazunori Miyata. A method of generating stone wall patterns. In *Computer Graphics, Proc. of SIGGRAPH 90*, pages 387–394. ACM, 1990.

[13] Mark Peercy, John Airey, and Brian Cabral. Efficient bump mapping hardware. In *Computer Graphics, Proc. of SIGGRAPH 97*, pages 303–306. ACM, 1997.

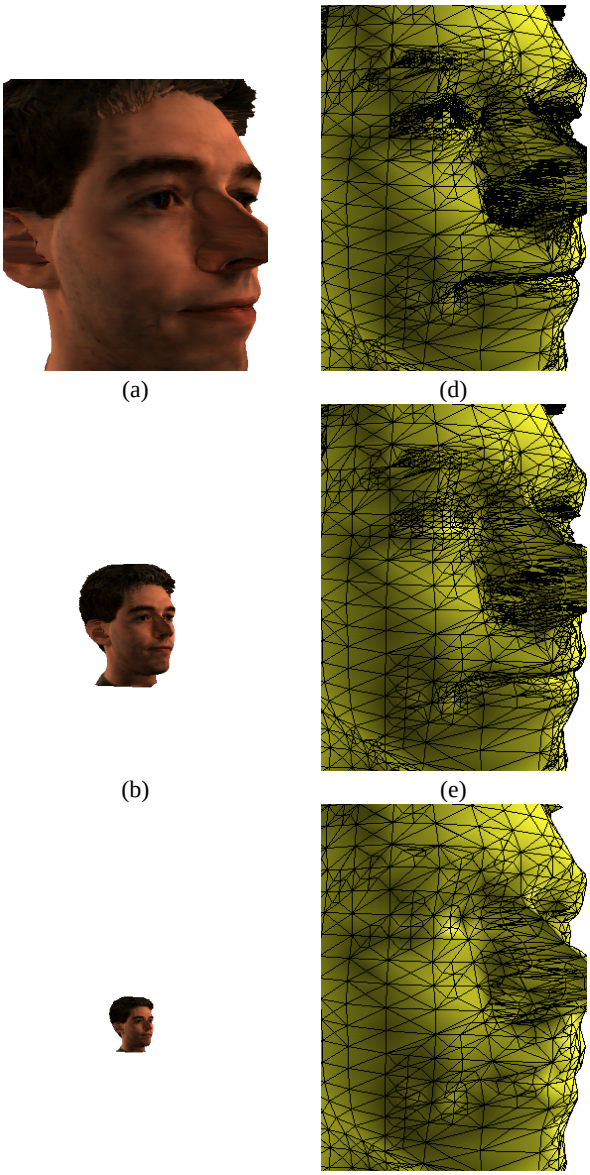
[14] Alyn P. Rockwood, Kurt Heaton, and Tom Davis. Real-time rendering of trimmed surfaces. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3):107–116, July 1989. Held in Boston, Massachusetts.

[15] Andreas Schilling. Towards real-time photorealistic rendering: Challenges and solutions. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 7–15, August 1997.

[16] Lance Williams. Pyramidal Parametrics. In *Computer Graphics, Proc. of SIGGRAPH 83*. ACM, 1983.

[17] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 299–308, July 1991.

[18] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide*. Addison Wesley, 1997.

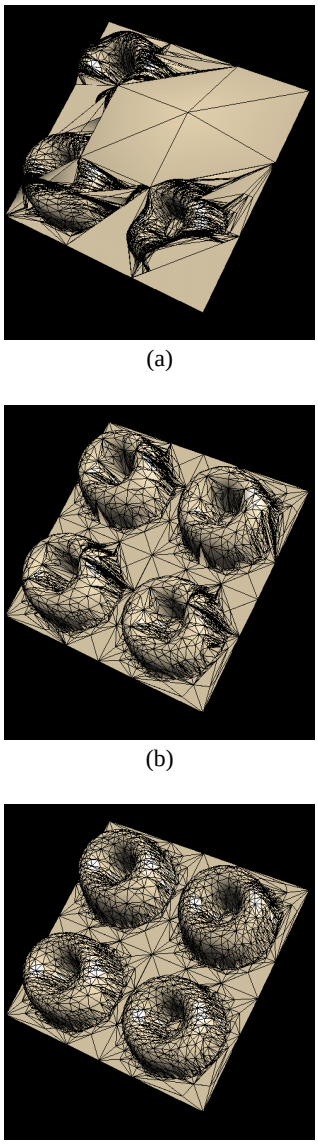


(a) (d)

(b) (e)

(c) (f)

Figure 9: Volker's head dataset. (a), (b) and (c) show distance from view plane, (d), (e) and (f) are mesh close-ups for corresponding images.

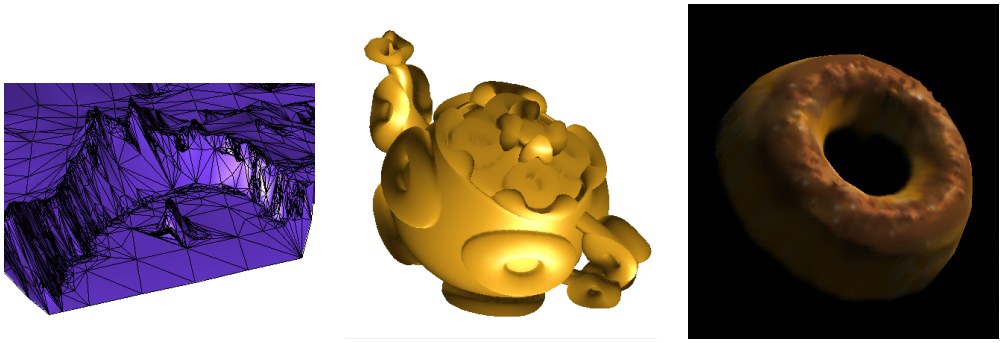


(a)

(b)

(c)

Figure 10: (a) Normal Test only. (b) Summed Height Test only. (c) Both tests.



(a) (b) (c)

Figure 11: (a) The Crater Lake. (b) A half donut displaced teapot (c) The half donut with texture and more surface detail added using another displacement map.