Paving Procedural Roads with Pixel Shaders

Jörn Loviscach

Hochschule Bremen
Flughafenallee 10
28199 Bremen, Germany
jlovisca@informatik.hs-bremen.de

ABSTRACT

Modern graphics hardware can be used to create procedural geometry. Our proposal details an optimized method to form roads and similar 3D objects by cookie-cutting them from slightly oversized polygons. The roads follow spline-like curves on a plane. The curves and their offset variants are cast into an approximated, implicit description. This can efficiently be evaluated within a pixel shader to discard pixels that are part of the oversized polygons but not part of the roads. Our method guarantees smooth geometry and smooth texturing. To achieve comparable results with roads formed from polygons in the usual way requires level-of-detail or similar mechanisms which not only complicate development and scene management, but also add load on the CPU.

Keywords

driving simulator, implicit curve, offset curve, pixel shader, clipping

1 INTRODUCTION

Roads are a prominent feature of virtual reality and gaming applications such as driving simulators. Many roads follow curved paths, in particular circles and spirals [AAS01], which are rendered with a large number of polygons. If this is not done, both the lateral borders of the roads and their textures such as medians show objectionable angles, see Figure 1.

Typical applications use large numbers of roads. To prevent a serious drop in the frame rate, these may not be rendered with a high polygons count. Thus, the number of polygons used has to be reduced for less visible or invisible roads or parts of them. This not only leads to additional development effort but also requires visibility estimation and a more sophisticated scene management to be done on the CPU.

Conference Proceedings ISBN 80-903100-7-9
WSCG '2005, January 31-February 4, 2005
Plzen, Czech Republic.

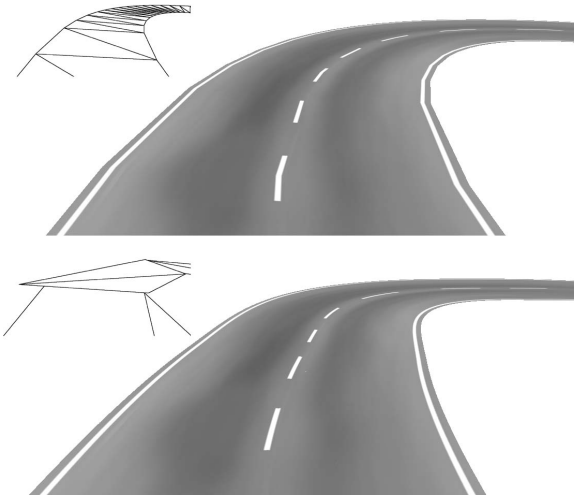

Figure 1. A conventionally built road shows angular artifacts (upper image). Our method yields smooth shapes and textures (lower image). The insets show the polygons used.

The main contribution of this paper is an efficient implicit description of fat planar curves: a centerline plus a family of offset curves. This implicit description is used to cookie-cut roads from large polygons using a pixel shader. These roads possess tangent continuous shape and texturing.

The input to our method is a set of anchor points—each equipped with a tangent direction—that represents the centerline (mostly marked by a median) of the road. Every two consecutive anchor points determine a road segment, which is to be treated separately. We assume that each segment lies in a plane. This is at least approximately valid for roads with slowly varying grade. Furthermore, we assume that no segment is strongly bent horizontally, so that its centerline can be parameterized using the projection onto the straight line that connects the two anchor points, see Figure 2.

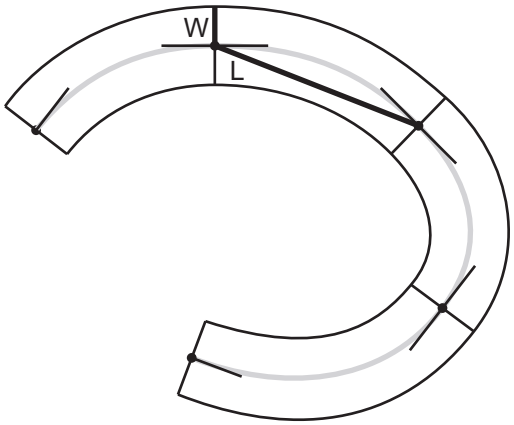

Figure 2. A road is divided into segments defined by start and end points on its centerline together with tangent directions.

Every road segment is covered with a quadrangle (to be rendered as two triangles under DirectX) computed from the road's width and the spline-like curve that forms the central line. A pixel shader is used to discard the pixels of the quadrangle that do not belong to the road. To this end, the `clip` instruction of the HLSL shading language is used. It translates to the `texkill` instruction of DirectX pixel shader assembler. Furthermore, the pixel shader assigns texture coordinates to the pixels. A mapping $\mathbf{x} \mapsto (u,v)$ is employed that ensures smoothness along every single road segment as well as tangent continuity at the transition from one segment to the next, see Figure 3.

All computations are offloaded from the CPU to the graphics hardware, excluding a short initialization routine to build vertex and index buffers. For optimization, the computation of all quantities that vary linearly is moved from the pixel shader into the vertex shader of the same (and only) rendering pass.

The proposed method does a substantial amount of work in the pixel shader. Roads close to the viewer incur a high computational cost, but are perfectly free from angular-looking defects. Distant roads, however, lead to only a small computational cost because they

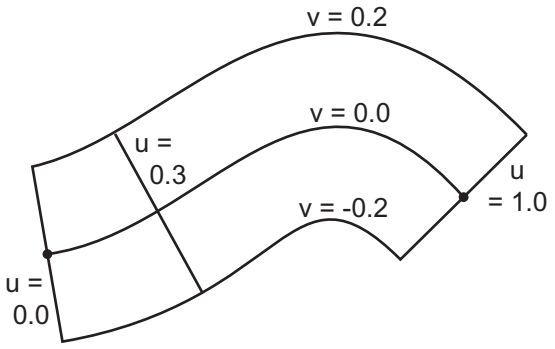

Figure 3. World space coordinates are converted to coordinates (u,v) along and across the road. These serve to define both geometry and texturing.

consist of few pixels in screen space. A large percentage of invisible segments of the road will be discarded already at the vertex level through frustum clipping. Only a low number of polygons is needed to construct the roads using the shader. Thus, the efficiency for distant roads is close to that of level-of-detail or frustum culling approaches.

This paper is structured as follows: In Section 2 we outline related work; Section 3 describes the formulation of roads as fat curves. How these can be evaluated using a graphics chip is covered in Section 4. Section 5 presents and discusses results; Section 6 gives a summary and points out directions for further research.

2 RELATED WORK

Vertex-based procedural creation of geometry on graphics cards has been studied much in recent years. For instance, it can be found in the curved-PN-triangle subdivision offered by current ATI graphics cards [Vla01]. Bolz and Schröder [Bol03] propose a vertex-based method to evaluate subdivision surfaces.

Due to increasing computing power and improved functionality, pixel-based instead of vertex-based procedural creation of geometry is now becoming a viable option. Some works have already addressed this topic.

Hirche et al. [Hir04] render per-pixel displacement maps on the graphics chip. To this end, they extrude prisms from the triangles of a mesh. They render these prisms with a complex pixel shader, which employs ray casting to evaluate the displacement map using four samples per ray. Kanai and Yasui [Kan04] evaluate per-pixel positions and normals of subdivision surfaces in a pixel shader and use the results to fill a vertex buffer to render the surface from. Loviscach [Lov04] uses curved fins along the silhouette of a mesh to smooth the outline visually. The fins are painted by a pixel shader onto quadrangles that are ex-

truded from the silhouette of the mesh inside a vertex shader.

Rose and Ertl [Ros03] draw wire frames onto simplified polyhedra. ATI's demo "Ruby: The Double Cross" [ATI04] employs pixel shaders to procedurally generate the ATI logo from lines and circles. This method does not actually produce geometry, but comes close in spirit.

In order to construct roads with a pixel shader, we use a parameterization that is related to offset curves: $v = 0$ is the centerline of the road; a non-zero v leads to an offset curve. Offset curves are a classic topic of computer graphics; for surveys see [Elb97] and [Mae99].

Most of the work done on offset curves is concerned with explicit representations. In contrast to that, we are interested in the inverse mapping from world space to parameter space, which may be compared to an implicit representation of offset curves. This can for instance be achieved with the distance function that maps every point to its distance to the original curve, a mapping that can be used, for instance, to find the medial axis transform.

Pottmann et al. [Pot02] study local quadratic approximations of the squared distance to a curve in the Frenet frame. They employ this approximation to generate offset curves with active splines. However, it does not seem straightforward to use these results here, in particular due to the non-global nature of the approximation.

3 ROADS AS FAT CURVES

For simplicity we only show the construction for the 2D case in which the central line starts at the origin $(x,y) = (0,0)$ and ends at $(1,0)$, see Figure 4. All other cases can be reduced to this by rotation and uniform scaling. Let the tangent direction at the origin be parallel to $(1,a)^{\mathrm{T}}$ and that at the end be parallel to $(1,b)^{\mathrm{T}}$.

Then we can construct the centerline as the graph of a function $y = f(x)$ with the following properties: $f(0) = 0 = f(1)$, $f'(0) = a$, and $f'(1) = b$. We choose f to be the cubic function that fulfills those requirements:

$$f(x) := x(1-x)^2 a - x^2(1-x)b$$

Given a point (x,y) near the curve, we want to find approximate values for the nearest position on the centerline (parametrized by x) and the signed distance from the centerline. Call these two values (u,v). The point (x,y) is a point on the road if and only if $0 \le u \le 1$ and $-W/L \le v \le W/L$, where W is half the road's width and L the distance between its anchor points (before scaling). Furthermore, (u,v) serve as curved texture coordinates on the road.

We assume that a, b, and y are close to zero so that there are no problems concerning the uniqueness of a nearest point on the curve. In Section 5 we will show how large these values may be chosen in practice.

To convert (x,y) to (u,v), we employ a basic idea from the theory of offset curves [Elb97]: The vector from (x,y) to the nearest point on the central line has to be perpendicular to a tangent vector to the curve, see Figure 4:

$$\begin{pmatrix} x-u \\ y-f(u) \end{pmatrix} \cdot \begin{pmatrix} 1 \\ f'(u) \end{pmatrix} = 0$$

This leads to

$$x - u + (y - f(u))f'(u) = 0. \tag{1}$$

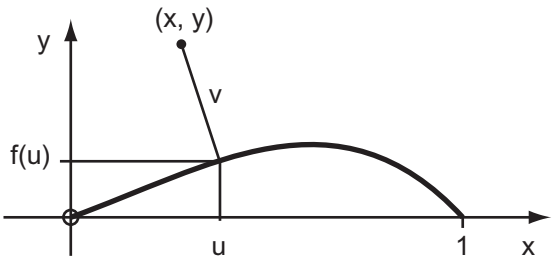

Figure 4. (x,y) is converted to (u,v) using the nearest point $(u,f(u))$ on the curve.

We are not going to solve Equation 1 (which in general is of degree five) precisely, but rather use it as a guidance to construct an approximately equal object with precisely identical properties regarding start point, end point, initial and final direction.

Note that $f(u) \approx 0$ for a curve which is only weakly bent. Furthermore, f' can be computed using a and b. This yields an approximation u_1 of u in Equation 1:

$$x - u_1 + y((3u_1^2 - 4u_1 + 1)a + (3u_1^2 - 2u_1)b) = 0 \tag{2}$$

This equation typically possesses two different solutions in u_1. We pick the solution close to x. This solution is guaranteed to exist for y, a, and b sufficiently close to zero. It can be written

$$u_1 = \frac{2\gamma}{\sqrt{\beta^2 - 4\alpha\gamma} - \beta}, \tag{3}$$

where

$$\alpha := 3y(a+b), \quad \beta := -1 - (4a+2b)y, \quad \gamma := x + ya. \tag{4}$$

We write the solution of the quadratic equation in the untypical form of Equation 3 to prevent a division by

zero when $y = 0$ and hence $\alpha = 0$. Note that $\beta < 0$ if the bending is weak and the width is small enough.

The points (x,y) with $u_1 = 0$ form the line through the start point $(0,0)$ perpendicular to the tangent of the curve at that point: From $u_1 = 0$ follows $x = -ay$. Similarly, the points (x,y), for which $u_1 = 1$, form a the line through the end point $(1,0)$ perpendicular to the tangent of the curve there.

A simpler approximation with the same properties would be

$$u_2 = \frac{x + ya}{1 + (a-b)y}.$$

However, it turns out that this approximation—concerning its overall shape—does not perform well for strongly curved paths.

Now v remains to be computed. If we had solved Equation 1 precisely, the signed distance v could be found through the dot product of a normalized vector perpendicular to the curve and the difference vector between the point (x,y) on the plane and the nearest point $(u, f(u))$, see Figure 4:

$$v = \frac{\begin{pmatrix} x-u \\ y-f(u) \end{pmatrix} \cdot \begin{pmatrix} -f'(u) \\ 1 \end{pmatrix}}{\sqrt{f'(u)^2+1}} \qquad (5)$$

Due to the dot product, this equation is robust under a small shift along the curve. Thus, it seems reasonable to use this equation in our framework with u_1 in place of u. This completes an efficient algorithm to convert a point (x,y) near the curve to curved coordinates (u_1, v).

Whereas the mapping $(x,y) \mapsto (u_1, v)$ is only approximate, it possesses the same features as the exact solution of Equation 1: On the lines $u_1 = 0$ and $u_1 = 1$ the mapping equals the exact solution, what is crucial for the continuous transition from one road segment to the next.

On top of that, the transition from one segment to the next is not only continuous, but also *tangent* continuous. To prove tangent continuity, one can compute the gradient of v with respect to x and y at $u_1 = 0$ and $u_1 = 1$ for arbitrary v using basic mathematics. It turns out that the gradient equals $(-a,1)^{\mathrm{T}}/\sqrt{a^2+1}$ and $(-b,1)^{\mathrm{T}}/\sqrt{b^2+1}$, respectively. All lines $v = \text{const}$ must run perpendicular to the gradient field. Therefore, they have a slope of a and b, respectively, at $u_1 = 0$ and $u_1 = 1$, what proves tangent continuity. This property of the construction does neither depend on the details of f nor on the approximation used to find u_1, as long as $f(0) = 0 = f(1)$, $f'(0) = a$, $f'(1) = b$, $u_1 = 0$ corresponds to $x = -ay$, and $u_1 = 1$ corresponds to $x = 1 - by$.

4 HARDWARE ACCELERATION

In the prototype, we have implemented the method using the following steps:

- Preprocessing:
  - Given a sequence of anchor points along the median of the road to be built, compute the initial and final slopes a and b for every segment using a Catmull-Rom spline that interpolates the anchor points. (The slope could also be defined arbitrarily.)
  - Create vertex and index buffers that describe one quadrangle per road segment.
- For every frame:
  - Draw the terrain.
  - Draw the quadrangles defined in the preprocessing step. Use shaders on them both to form the road through pixel clipping and to map a texture onto it.

Each quadrangle is chosen such that it covers the corresponding road segment completely with not much excess, see Figure 5. To create quadrangles with minimum area reduces rendering time. This construction employs T-junctions (see inset in the lower part of Figure 1), which may be objectionable in other circumstances. For a discussion see Section 5.

In order that the segments fit together, two of the sides of a quadrangle have to run through the start and end points, respectively, of the centerline, perpendicular to the corresponding tangents, see Figure 5. The two other sides of the quadrangle are parallel to the straight line $y = 0$ that connects the start and the end point of the centerline. To position these two sides, we look for extremal values of f on $[0,1]$ by solving the equation $f'(u) = 0$, which in general is quadratic. If, for instance, there is a maximum at $u = u_+$, the upper side has to be shifted upward to $y = f(u_+) + W/L$.

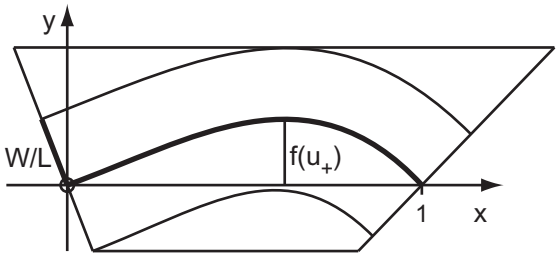

Figure 5. Every road segment is covered by a quadrangle.

Note that this shape saves the test whether $0 \le u_1 \le 1$, because this is automatically true for any point on the

quadrangle: Along one of its sides u_1 equals 0, along one other side it equals 1. Thus, only $-W/L \leq v \leq W/L$ remains to be checked to determine if a point lies on the road.

For each of the four vertices of such a quadrangle we store the following attributes in the vertex buffer:

- 3D position
- xy position in the rotated and scaled system according to Section 3
- a, b, and the distance L between the anchor points
- u_{Start}, u_{Range}

The distance L is needed to adapt the road width (which is transmitted as a constant parameter), because the xy position is scaled to $x \in [0,1]$. The values u_{Start} and u_{Range} are used to shift and rescale u, the texture coordinate along the road. Both are determined from arc length such that the textures of the road segments fit together seamlessly.

The data a, b, L, u_{Start}, and u_{Range} are identical for all vertices of one quadrangle. Therefore, we use the Vertex Stream Frequency Divider offered by DirectX 9.0 to save memory bandwidth for this subset of the vertex attributes.

The value α, β, and γ of Equation 4 depend linearly on the position of a pixel inside a triangle. For efficiency, we use a vertex shader to compute them per vertex, and rely on the automatic linear interpolation applied by the graphics chip to all values that are transmitted from the vertex shader to the pixel shader. The pixel shader then evaluates Equations 3 and 5.

For the implementation we chose Managed DirectX 9.0c using the language C# and Microsoft's Effect framework with HLSL. The vertex and the pixel shader compile to 23 and 35 instructions, respectively, of Shader Model 2.0. All computations are done using 16 bit floating point precision instead of the regular 32 bit floating point precision without visually objectionable roundoff errors.

In a typical virtual reality or gaming setting, the geometry of buildings and terrains can be much more angular than that of roads: Most buildings possess rectangular forms by construction; terrains can be covered with complex textures that help to hide large polygons. Smooth roads may, however, not be combined with a coarsely-tessellated terrain in a straightforward manner: The roads would be cut off in angular patterns.

To prevent this, the terrain in the vicinity of the road is composed of large level polygons, see Figure 6. Another possibility would be to introduce ditches. We render the terrain before the roads and leave a visually unnoticeable small height gap between the road and the terrain below to prevent z-fighting.

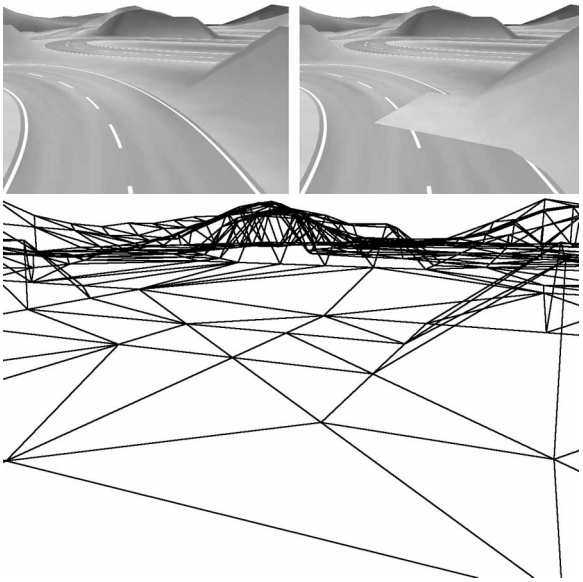

Figure 6. Intersections between the roads and the terrain may reveal the coarse tessellation of the latter (top right). Hence, we create level geometry along the roads (bottom).

5 RESULTS. DISCUSSION

Even for relatively large values of a, b, and W/L the approximate mapping $(x,y) \mapsto (u_1, v)$ yields useful results, and the quadrangle fits closely, see Figure 7. In our experiments, we found no visually objectionable deviations as long as $|a| \leq 1$ and $|b| \leq 1$, and furthermore $|W/L| \leq 0.3$ if a and b are of different sign and $|W/L| \leq 0.2$ if they are of same sign. This range allows strongly curved segments, see Figures 8 and 9.

In situations with strong bending such as that of Figure 9 the viewer may realize that the u_1 coordinate used for texturing deviates from arc length parameterization. This difference could be diminished through a corrective term. With typical road textures, however, this is not necessary.

To fit the quadrangles tightly around the road, we employ geometry with T-junctions. Thus, roundoff may lead to pixel-wide gaps between two quadrangles. However, in our experiments such defects did not turn up. One may also argue that the number of vertices could be cut by half by joining every two neighboring vertices on each side along the road. But this would enlarge the area of the quadrangles and thus lead to more invocations of the pixel shader. Furthermore, as described in Section 4, every vertex contains the values of x and y in its local coordinate frame. A shared

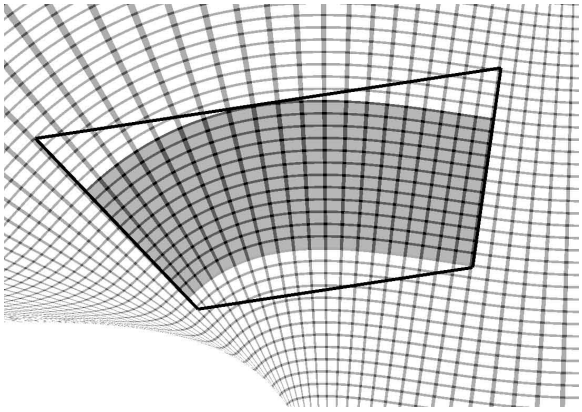

Figure 7. The parameterization of a road segment ($a = 0.7$, $b = -0.3$, $W/L = 0.2$) and the quadrangle used for rendering show that the approximation in Eq. 2 does not lead to easily recognizable errors.

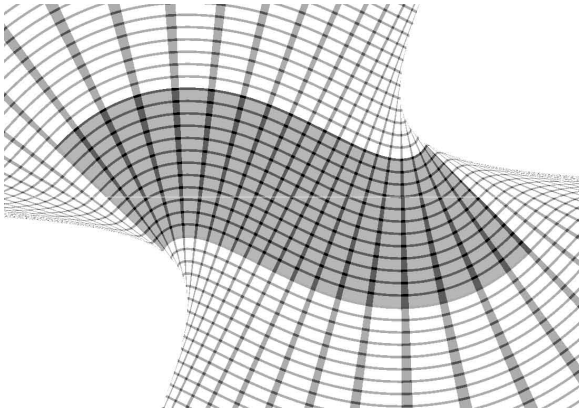

Figure 8. Strongly bent curves such as for $a = 1$, $b = 1$ must not be too wide. Here, $W/L = 0.2$, which is the allowable maximum for these values of a and b.

vertex would have to be equipped with two sets of these data—one for the previous quadrangle, one for the next. It is hard to see how the shader could switch between both sets.

For the speed benchmarks we used an Nvidia GeForce FX 6800 graphics card in a PC equipped with an Intel Pentium-4 processor running at 2.5 GHz. The rendering was done in 1280×1024 full screen mode without vertical synchronization.

Because roads are typically viewed under a very oblique angle, textures have to be filtered anisotropically. In our experiments, a setting of 4 for the maximum degree of anisotropy proved to be sufficient, see Figure 10.

To study the scaling behavior we used a base scene, see Figure 11, as a building block to create seven scenes

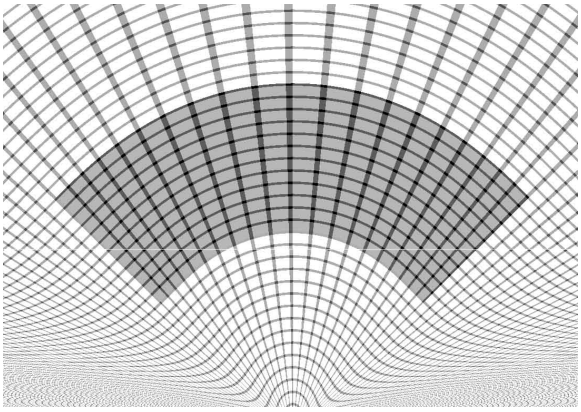

Figure 9. A quarter circle ($a = 1$, $b = -1$, $W/L = 0.2$) is approximated with a peak error of 25 percent, which, however, is not immediately apparent.

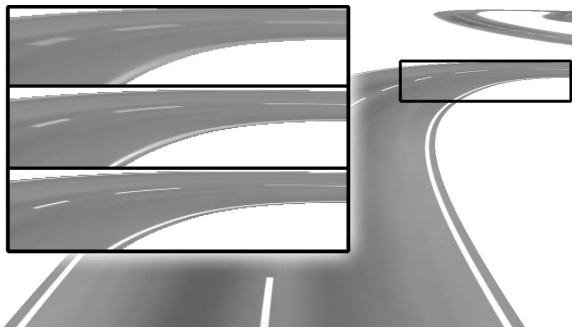

Figure 10. To avoid noticeable blurring, we set the maximum anisotropy level of texture filtering to 4. The inset on the left shows the portion outlined on the right with levels 1, 2, and 4 (top to bottom).

of different complexity ranging from one copy of the base scene to 25×25 copies arranged side by side in a rectangular pattern. The road of the base scene is composed of 97 segments. In addition, we created a terrain consisting of 2868 triangles, rendered before the road. We used a field of view of 45° and a far plane distance of 1.5 times the longer side length of the base terrain.

To compare the shader-based solution with a purely polygonal construction, we used the software package Maxon Cinema 4D to create a Catmull-Rom spline curve from the anchor points. (Note that the centerline of the road generated by our method is no such curve, but a visually close approximation.) The spline was extruded into a road, which was stored inside an .x mesh, imported and rendered inside our software prototype.

To have a basis for comparison, we generated a set of five differently tessellated versions using the

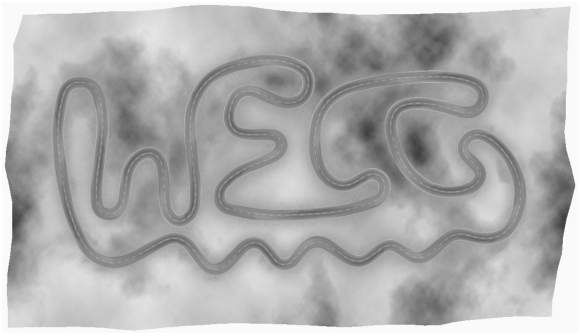

Figure 11. The base scene for the benchmark comprises a road defined by 97 anchor points and tangents.

curvature-adaptive setting of Cinema 4D with threshold angles of 1°, 2°, 5°, 10°, and 20°, respectively, which led to polygonal versions of the road consisting of 5374, 2972, 1392, 772, 444, and 256 triangles, see Figure 12. Only the highest one of these resolutions could warrant that the shape and the texture of the road looked perfectly smooth from viewpoints such as that of Figure 12.

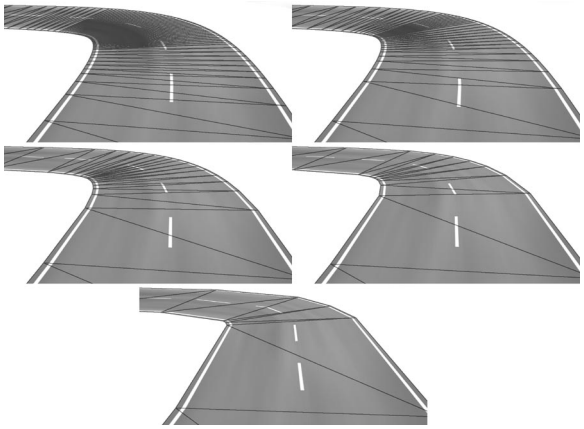

Figure 12. To compare the cookie-cutting method with other approaches, we used five different tessellations at threshold angles 1°**,** 2°**,** 5°**,** 10°**, and** 20° **(from top to bottom).**

Whereas per-pixel procedural geometry in itself is more expensive than standard polygons, the proposed approach may outperform roads rendered from polygons in scenes with high complexity, see Figure 13. This is mainly due to the strongly reduced amount of polygons to be discarded during view frustum clipping. To achieve a similar effect, a purely polygon-based approach may switch to the "5°" or a coarser version based on distance (level of detail) or use some sort of hierarchical frustum culling.

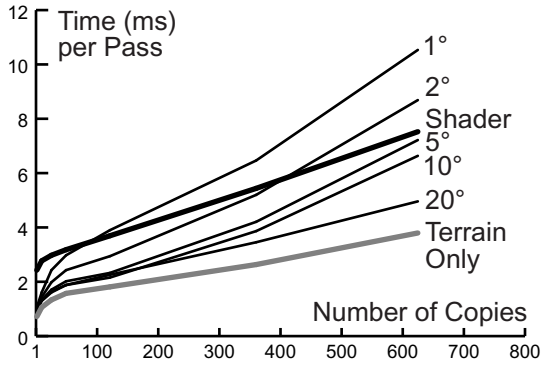

Figure 13. The benchmarks compares the rendering times for our shader-based method, standard renderings with varying degrees of tessellation, and the terrain without the road.

In principle, our solution should also benefit greatly from early-z optimization. If the terrain is drawn before the roads, the graphics chip would be able to cull all pixels of roads that are hidden beneath terrain geometry. This could reduce the workload of the pixel shader drastically. However, currently the cookie-cutting approach (i. e., use of the `texkill` instruction in the pixel shader) will disable early-z optimization of typical graphics cards [Rig02, Nvi04].

6 CONCLUSION. OUTLOOK

We have presented a method to generate roads and similar 3D objects procedurally with a pixel shader. This approach generates smooth shapes and textures with little effort for initial setup and no runtime scene management overhead. In contrast to that, level-of-detail switching would have to involve countermeasures against popping artifacts or intersections between coarse versions of roads and the terrain.

For roads close to the viewer, the shader-based method leads to a perfect look but adds a noticeable computational load on the graphics chip. For distant roads and scenes of high complexity, the performance approaches that of the standard method of tessellating objects into fine triangles. Given the fast performance growth of graphics chips as opposed to that of CPUs, this may also become true for scenes of medium complexity in the near future.

Most applications of our method will need to combine the curved roads with intersections etc. The latter can be built using standard polygon-based geometry. The transition between such a crossing and a road cookie-cut from polygons can be constructed easily because the road conforms to precise boundary conditions concerning width and direction.

It is straightforward to add sidewalks to the method. One can use a quadrangle shifted upward by the sidewalk's height and cookie-cut this at the corresponding v values. The size of the quadrangle used can be adapted. However, sidewalks need curbs. Their surface is not contained in a plane, so that a different method than the one described here is needed if they are to be generated procedurally.

We have treated only level roads. If its grade varies only slowly, a road may be constructed from segments that form small angles to each other in the vertical direction. In addition to curvature in the vertical direction one may also try to reproduce such features as superelevation, which means a rotation about the centerline.

It seems plausible that smoothly bent tubes can be generated by a pixel-based method that is similar to the one described. However, such a method would have to employ billboard-type pseudo-geometry, which always faces the viewer. Furthermore, it would have to address shading, too. To this end, normal vectors can easily be derived from the curved coordinates.

7 ACKNOWLEDGMENTS

The author wishes to thank two of the anonymous reviewers for providing detailed comments, which proved very helpful for clarification.

8 REFERENCES

[AAS01] American Association of State Highway and Transportation Officials. A Policy on Geometric Design of Highways and Streets. AASHTO, 2001.

[ATI04] ATI. Making of Ruby, http://www.ati.com/developer/SIGGRAPH04/ MakingOfRuby_Slides.pdf, 2004.

[Bol03] Bolz, J., Schröder, P. Evaluation of Subdivision Surfaces on Programmable Graphics Hardware. Submitted for publication, http://www.multires.caltech.edu/pubs/GPUSubD.pdf, 2003.

[Elb97] Elber, G., Lee, I.-K., Kim, M.-S. Comparing Offset Curve Approximation Methods. IEEE Computer Graphics and Applications 17(3), pp. 62–71, 1997.

[Hir04] Hirche, J., Ehlert, A., Guthe, S. Hardware Accelerated Per-Pixel Displacement Mapping. Proc. of Graphics Interface 2004, pp. 153–158, 2004.

[Kan04] Kanai, T., Yasui, Y. Per-Pixel Evaluation of Parametric Surfaces on GPU. Surface Quality Assessment of Subdivision Surfaces on Programmable Graphics Hardware. Proc. Int'l Conf. on Shape Modeling and Applications 2004, pp.129-136, 2004.

[Lov04] Loviscach, J. Sillhouette Geometry Shaders. In: Engel, W., ed., ShaderX^3: Advanced Rendering With DirectX and OpenGL, Charles River, pp. 49–56, 2004.

[Mae99] Maekawa, T. An Overview of Offset Curves and Surfaces, Comp. Aided Design 31, 165–173, 1999.

[Nvi04] Nvidia GPU Programming Guide, Version 2.2.0, http://developer.nvidia.com/object/gpu_programming_guide.html, 2004.

[Pot02] Pottmann, H., Leopoldseder, St., Hofer, M. Approximation with Active B-Spline Curves and Surfaces. Proc. Pacific Graphics 02, pp. 8–25, 2002.

[Rig02] Riguer, G., Performance Optimization Techniques for ATI Graphics Hardware with DirectX 9.0, Revision 1.0, http://www.ati.com/developer/dx9/ATI-DX9_Optimization.pdf, 2002.

[Ros03] Rose, D., Ertl, T., Interactive Visualization of Large Finite Element Models, Workshop on Vision, Modelling, and Visualization VMV '03, pp. 585–592, 2003.

[Vla01] Vlachos, A., Peters, J., Boyd, C., Mitchell, J. L. Curved PN triangles. Proc. 2001 Symp. on Interactive 3D Graphics, pp. 159–166, 2001.