RAY TRACING OF PARAMETRIC SURFACES BASED ON ADAPTIVE SIMPLICIAL COMPLEXES

Franz Duckstein, Reiner Kolla
Computer Science Department – Julius Maximilian University
97070 Würzburg, Germany
{duckstei,kolla}@informatik.uni-wuerzburg.de

ABSTRACT

Objects defined by parametric surfaces are a compact representation delivering continuous defined accuracy. For the sake of speed, a discretization towards triangles is inevitable, which can be done adaptively to the projective properties. Simplicial complexes suit perfectly to store the adaptive discretization. Based on adaptive simplicial complexes we propose a modified ray tracing algorithm. The algorithm steers the scene discretization in order to deliver the required accuracy. Invisible object areas gain less accuracy, which increases the time available for processing visible parts. With an appropriate hardware support we hope to get ray tracing like quality for dynamic complex scenes in real time.

Keywords: Virtual Reality, Parametric Surfaces, Adaptive Tessellation, Ray Tracing, Radiosity

1 INTRODUCTION

The creation of virtual reality within the computer graphic is still an open problem. Virtual reality demands complex scenery, high detail per frame and animation within real-time. These three aims are contradictory. All modern methods try to find a trade off between these demands. Dependent to the desired application the criteria are weighted differently to find a solution. Usually this weighting is fix, because it is done as a first step in the design. There is less flexibility to handle the changes within dynamic scenes. A flexible weighting of calculation effort during every step of calculation seems to be advantageous. How can this be done? The scene representation should be able to deliver accuracy on demand, without sticking up the system by its complexity. The algorithm used to calculate a frame should be able to create enough realism, so light/shadows, reflection and refraction are required.

Scene Representation - the polygonal scene representation is omnipresent within the 3D real-time computer graphic. The high render capacity delivered by modern render hardware often reveals the bus between *cpu* and graphic subsystem (*gss*) as a bottleneck. A dedicated graphic bus from *cpu* to *gss*, e.g. Intel's *agp*, with a high capacity is no general solution. Huge polygons numbers require also a sufficient geometric preprocessing from the *cpu* itself. The Amd 3D-now extension tries to reduce this problem by a floating point extension. Level of detail management is a well known method, saving communication and render cost [Funkh93, Cohen96]. Every scene object is given with several fixed representations. As a disadvantage, additional memory is needed to hold the representations. Complex objects (> 10k polygons) deny the application of fixed *lod*. A solution to a subset of this objects, 2D manifolds, was brought by polygon meshes [Hoppe96, Luebk97]. They collapse adjacent polygons to one coarser polygon. By iterated collapsing steps, a hierarchical simplified representation is found. During the display process the right granularity for every object area is calculated dynamically. The close or contour parts of objects will be fine resolved, distant or smooth object parts can be kept low resolved. With *simplicial complexes* this principle is applicable to polygons sets with complicated topology [Popov97].

Disadvantageous to all this methods is a detail limit, caused by the polygonal representation base. Note that these complex polygon sets are often created by the evaluation of parametric surfaces, e.g. b-splines. The polygon as an interface between the host and a sophisticated *gss* seems to be unsuited. We prefer the parametric object description as an interface. There is few data about position, movement and a surface function to transmit, while accuracy can be chosen individually.

Picture Calculation - polygonal scenes are usually displayed by polygon rendering. The produced realism is founded on the scene complexity, texturing and interactive frame rate. Additional lighting or reflection calculations are expensive and limited. Ray tracing delivers the required realism. But within complex (polygonal) scenes, the run time demands are beyond real-time limits. To cope this problem, the overall scene complexity must be kept low. We investigate a modified ray tracing algorithm to control the object accuracy within every phase, during visibility, shadowing and reflection calculations. The major principle within this algorithm is to introduce accuracy on demand within *every* step of the calculation. Every ray traced to the scene is used to control the further scene refinement, but without voiding previous ray results. As a consequence, we keep the overall complexity low.

Our first step was to prove the algorithm capabilities. We implemented a sequential version, but the main principles can be transfered to parallel. The paper is structured as follows. In the next section we explain the object discretization. An adaptive refinement implies the need of an appropriate storage structure. The section afterwards explains the gearing between the adaptive and incremental discretization with the modified ray tracing algorithm. To deliver correct results, the intersection-test of rays with scene elements must be scheduled. At the end, we show some practical results and statistical evaluations, which are very promising.

2 OBJECT REPRESENTATION

Parametric surfaces can deliver infinitely variable accuracy combined with small fixed storage requirements. So they are well suited as a base to build up complex scenes. For the display process within polygon rendering, a discretization towards triangles is needed. The tessellation is usually done constant or view-driven [Rockw89]. Within ray tracing, only a few parametric surfaces like quadrics can be handled with the necessary intersection-test for a ray. For other kinds, like *nurbs*, the ray intersection-test is difficult. It can be done iterative, which is time expensive. The intersection of ray and triangle is cheap to calculate. Therefore a discrete approximation by triangles is mostly preferred. The used principles are well studied and common practice. As a disadvantage within ray tracing, the effort per ray is linear to number of elements within the scene. So the tessellation of parametric surfaces is always a trade off between needed accuracy and run time requirements. Usually the tessellation granularity is calculated in a preprocessing step. And without the knowledge of the later ray results only a geometric error function can be used. For example the *Hausdorff distance* [Klein96] between discrete and parametric surface, or the discretization is simply controlled by projective properties [Synde87].

PARAMETRIC SURFACES

Parametric surfaces have a long tradition within computational geometry and computer graphics. There are three major kinds: quadrics, interpolation surfaces, e.g. *splines*, and approximation surfaces, e.g. *nurbs*. The analytical and geometrical properties are described in [Foley90], [Encar88]. Within our application, these properties are considered to be transparent as far as possible. The parametric function is used to deliver sample points, which one is controlled by a conform interface. Regarding to parametric surfaces and their discrete approximation we use in our paper the concepts and notations out of [Berto95, DeFlo97, Jaeni94].

A *parametric surface* is the image of a continuous function $\Phi : \Omega \to \mathcal{R}^3$, where $\Omega \subseteq \mathcal{R}^2$ is a compact domain. The compact notation $\Phi(\Omega)$ of a parametric surface specifies the surface function and its domain. The *physical space* $\mathcal{R}^3$ embeds $\Phi(\Omega)$, the *parameter space* $\mathcal{R}^2$ contains the domain Ω of Φ.

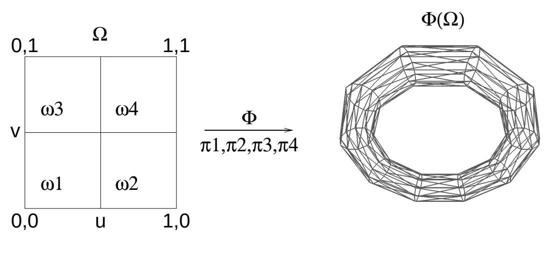

Figure 1: A parametric torus surface $\Phi(\Omega)$, divided into 4 injective patches $\pi_1, \pi_2, \pi_3, \pi_4$ within the canonical parameter space $\Omega = [0,1]^2$

Injectivity of Φ on Ω is required. This allows the discretization to be realized by a domain decomposition of Ω. If the function Φ is only injective to the interior of Ω, not on its borders, the division of $\Phi(\Omega)$ into injective *patches* $\pi_i(\omega_i)$ is in most cases sufficient. If $p \in \Phi(\Omega)$ is not a finite set of points in its inverse image, the function Φ itself must be transformed. As a further restriction, we assume Ω in the *canonical* form $[0,1]^2$, without trimmed regions. This means Ω is a joined domain. If Ω is splitted, the subregions $\omega_1, \ldots, \omega_k$ must still be rectangular areas. See Figure 1 as an example.

SIMPLICIAL COMPLEXES

Simplicial complexes are a common way to describe fixed or adaptive discrete multi-resolution boundary representations. They deliver the required topological order with moderate effort [Berto95],[Popov97]. By the support of additional parametric surface vertices they are capable to build up the geometric and topological complexity incremental, starting from a coarse initial representation.

A d-simplex σ within the $\mathcal{R}^n$ is defined by the convex hull $s(\nu_0, \ldots, \nu_d)$ of $d+1$ vertices, which must define d linear independent vectors. The order of σ is d. Any s-simplex σ' with $0 \leq s \leq d$, generated by a subset of $s+1$ vertices of σ is called an *s-face* of σ. A *proper face* of σ is given for $s < d$.

A finite set Σ of simplices within the $\mathcal{R}^n$ is called a *simplicial complex*, if the following condition holds:

- for each simplex $\sigma \in \Sigma$ also all faces of σ are contained within Σ
- for each $\sigma_i, \sigma_j \in \Sigma$ is the cut $\sigma_i \cap \sigma_j$ empty or a face common to both

The order of Σ is the maximum order of its simplices. We work with 2-simplices (triangles) and 2-complexes (triangulation) in the $\mathcal{R}^3$. As a further requirement, each vertex is oriented by a surface normal. If every 0-simplex (vertex) $\sigma \in \Sigma$ holds the condition $\sigma \in \Phi(\Omega)$, we say $\Sigma_{\Phi(\Omega)}$ or short Σ is a discrete representation of $\Phi(\Omega)$. The global error between accurate and discrete form is noted by $E(\Sigma)$, the local error within one simplex by $E(\sigma)$.

The *topological realization* $|\Sigma|$ is defined as $|\Sigma| = \langle\Sigma\rangle = \cup_{s \in \Sigma} \langle s \rangle$. If the condition $\forall$ s-simplex $\sigma' \in \Sigma, s < d$ exists a d-simplex $\sigma \in \Sigma$ with σ is a proper face of σ' holds, a d-complex Σ is called *regular*. Within our paper we always consider regular 2-complexes, which means $|\Sigma|$ is already defined by the union of all 2-simplices out of Σ.

To allow iterative refinement during the picture calculation, an hierarchical triangle mesh would be sufficient for closed parametric surfaces. But regarding to parametric surfaces with *genus* $>$ 0 a single triangle mesh is not the appropriate topological representation, to start from an initial coarse representation. Regular simplicial complexes, iteratively refined from the coarsest form Σ^0 to the final Σ^f form, represent always the same topological type[1]. Therefore we formally use a simplicial complex Σ as discrete approximation of $\Phi(\Omega)$, while a single $\Sigma_{\pi_i(\omega_i)}$ is treated as a hierarchical triangle mesh within our algorithm.

RESTRICTIONS

If the parametric surface $\Phi(\Omega)$ is split up into injective patches $\pi_1(\omega_1), \ldots, \pi_k(\omega_k)$ the parametric function $\Phi = \{\pi_1, \ldots, \pi_k\}$ may change. Maybe, because the original Φ was transformed per patch to be injective. Or the function Φ is an assembled function. As a consequence, there is a need to describe the relation between neighboured patches [DeFlo97]. Each rectangular parameter subspace ω_i has four border sides. Within a closed boundary representation each side must be linked to *one* another subspace. Otherwise up to three border sides maybe unlinked. Self links are not allowed, to grant injectivity. In addition, a link type must be defined. We use three types, no continuity, C^0 continuity and C^1 continuity. An example is illustrated in Figure 2. For the later use of *nurbs* patches the introduction of higher continuity maybe required.

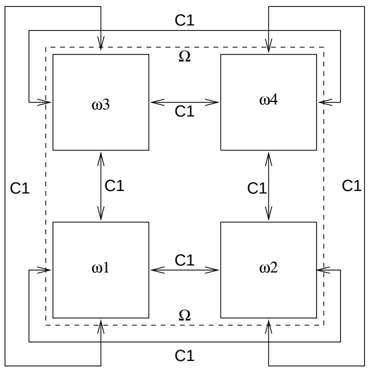

Figure 2: Patch links and link type continuity C^1 for the torus shown in Figure 1. Note, that all borders are linked (solid object boundary)

3 VERTEX RAY TRACING

Realistic pictures need to simulate realism as far as possible. Ray tracing calculates this realism by simulating the optical law's per ray. As a drawback, the complexity per ray is $O(n)$. And every primary ray can spawn a ray tree of $(m+1)(2^d-1)$ rays[2] to calculate recursive reflection, refraction and shadowing [Foley90]. The combination of ray

[1] they are homeomorphic

[2] d is maximum ray depth, m number of light sources

tracing and parametric surfaces is more difficult. By a discrete hypersurface representation, we can use the cheap ray-triangle intersection test, but n increases dramatically.

PRINCIPLES

Optimized ray tracing algorithms like *beam tracing* [Heckb84] or *pencil tracing* [Shiny87] produce good and fast results. They exploit coherence and continuity within the scene in order to reduce the calculation effort. But the scene's complexity is still a constant factor. We use an initial coarse scene representation and try to steer the scene complexity as far as possible on demand. The major principles are:

· compact description, also for complex objects
· find out *constant areas* within the picture, approximate them coarse
· find out *varying areas* within the picture, approximate them fine
· exploitation of *constant areas* reduces effort
· recognition of *varying areas* delivers accuracy

The classification of areas is dependent from the object's visibility, orientation and parametric function:

constant areas: parametric function is flat, and equal influences by light, reflection and refraction

varying areas: parametric function is varying or different influences by light, reflection or refraction

For a single 2-simplex (triangle) this problem appears as follows. Every simplex is continuously trying to decide, if it is a proper approximation of the parametric surface, regarding to the parameter space of ω it covers. This is done by rays, which are emitted from the vertices to gain information. The ray results control the refinement of the emitting vertices[3]. Different ray results within the three edges of a simplex indicate regions of high information. In addition, rays by refinement within this region are the consequence. To integrate geometric object properties, like surface curves, and the view-dependent properties, like frontal or side view, the simplex vertex data (position, its projection and the normal) must be considered. To simplify the model we incorporate the ray results as a vertex property. Here the algorithm skeleton:

```
forall Σ ∈ {Σ_1^{j_1}, ..., Σ_n^{j_n}} do parallel
{ Emission; Result; Request }
until nothing to do

Emission: forall vertices ν ∈ Σ^j do parallel
  if (not already emitted)
    emit visibility ray from ν
  if (not already emitted and ν visible)
    emit secondary rays from ν

Result: forall finished rays do
  add r-result to property of emitting ν
  forall σ_k ∈ σ(ν)
    if (vertex properties of σ_k are different)
      refine σ_k to σ_{k,1}, σ_{k,2} (Σ^j → Σ^{j+1})
    forall finished σ_k ∈ Σ^j
      repeat deferred ray requests

Request: forall incoming ray r do
  Σ' = Σ^0
  forall σ_k ∈ Σ' do
    if (intersect(r,hull(σ_k)) and σ_k is not finished)
      σ_k defers r
    else if (intersect(r, σ_k) and σ_k is refined)
      Σ' = (Σ' \ σ_k) ∪ σ_{k,1} ∪ σ_{k,2}
    else if (intersect(r, σ_k) and σ_k is finished)
      if (σ_k is visible)
        r notifies a result
      else
        σ_k is visible now, recheck properties
        r is deferred until σ_k is properly refined
```

Vertices: The set of $\{\sigma_k : \nu \text{ is a vertex of } \sigma_k\}$ defines $\sigma(\nu)$. A vertex ν is **visible**, if the ray to the viewer is not blocked, or one of the $\sigma_k \in \sigma(\nu)$ was hit by a reflection or refraction ray.
Simplices: A simplex σ_k is called **finished**, if it was **refined** to $\sigma_{k,1}, \sigma_{k,2}$ by a split, or its vertices properties are equal. Two vertices ν_1, ν_2 have **different properties**, if the projective distance is greater than a *coarse*-pixel limit or the normal cone of ν_1, ν_2 exceeds an angle limit or the emitted rays hit different targets. Otherwise they are **properly refined**. If the projective distance is below a $fine$-pixel limit[4] the two vertex properties are always equal. A simplex is also finished, if all its vertices are invisible.
Rays: A **visibility ray** is sent from vertex ν to the viewpoint. A **secondary ray** is for lighting, reflection and refraction, dependent to the material properties. A ray is **finished**, if an intersection with a finished simplex without sons occurs and the ray was sent to the viewer or a light source. The refraction and reflection rays must first notify all results and find the closest hit before they can finish.

All finished simplices without sons, which are not properly refined, can proceed to refine if they are hit by secondary rays. This is required to avoid artefacts, e.g. coarse shadow edges by unrefined

[3]one vertex can be assigned to several adjacent simplices
[4]coarse/fine-pixel and cone limits can be varied

backward surfaces. In order to speed up the intersection calculations, each simplex has a surrounding hull (sphere). The intersection-test of ray and simplex is only done, if the sphere is hit by the ray. This principle is also applied to unfinished simplices. To avoid artefacts, the sphere radius must be chosen with respect to the parametric function. After the iterative algorithm has finished, all visible properly refined simplices are displayed. The ray results received by the vertices are stored as a property, e.g. the color returned by secondary rays. In order to deliver a closed surface, each simplex interpolates its vertex colors during the display process by *gouroud* shading.

INCREMENTAL REFINEMENT

The calculation is started with an initial refinement of all parametric injective patches. The initial discrete representation per patch π consists of four simplices, calculated by a symmetric division of its parameter space ω. The union of all initial discrete representations builds Σ^0. See Figure 3 as an example for the refinement process. Vertex properties are compared per edge. By comparing the edge differences, a decision is made, which edge is to be split. An edge split is done in patch parameter space ω. Let the edge vertices ν_i and ν_j build the current edge ξ. The new parametric surface point $\nu_{i,j}$ is derived as follows. Each vertex ν stores it parameter space position u_ν, v_ν. For $\nu_{i,j}$ we set $u_{\nu_{i,j}} = (u_{\nu_i} + u_{\nu_j})/2$ and $v_{\nu_{i,j}} = (v_{\nu_i} + v_{\nu_j})/2$ and calculate $\nu_{i,j} = \pi(u_{\nu_{i,j}}, v_{\nu_{i,j}})$. The new vertex is taken to define two new edges $\xi_1 = \overline{\nu_i \nu_{i,j}}$ and $\xi_2 = \overline{\nu_{i,j} \nu_j}$. Together with the unsplit edges they build two new simplices. If an edge is common to two simplices, both are refined.

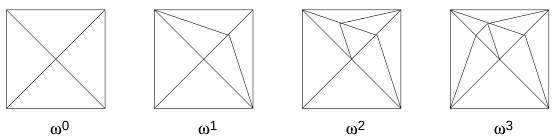

Figure 3: Refinement of patch parameter space ω from initial state ω^0 to ω^3 by three successive splits. Each simplex in parameter space correlates to a simplex in physical space $\Sigma_{\pi(\omega)}$

VISIBILITY

The algorithm sketch is an unified scheme. Within our implementation we split up the scheme into three different stages. There are two reasons to do so. First, the incremental refinement by ray results is inherent susceptible for deadlocks. Separate stages alleviate this problem. Second, the overall load in a sequential algorithm is reduced, because the number of deferred rays is kept low.

[5] patches have always genus 0

The first stage is the *visibility* calculation towards the viewer. Per vertex a visibility ray is emitted. A simplex is visible, if at least one of its vertices is visible, and it is perhaps visible, if its vertices are blocked by different patches[5]. Otherwise its regarded as invisible towards the viewer. All (perhaps) visible simplices are refined, if required by the geometric and projective properties. A simplex without the need of further refinement is preliminary finished. But how to decide a vertex as visible, if all or some objects are only coarse approximated. This problem is solved by deferring rays. Each simplex, which is not finished, defers the ray processing, until it is finished or refined. When a simplex is refined, the derived simplices inherit its deferred rays. When a simplex is finished, all its deferred rays are processed. So, the scene is refining itself from front to back. Hidden areas are not refined. Figure 4 shows an example.

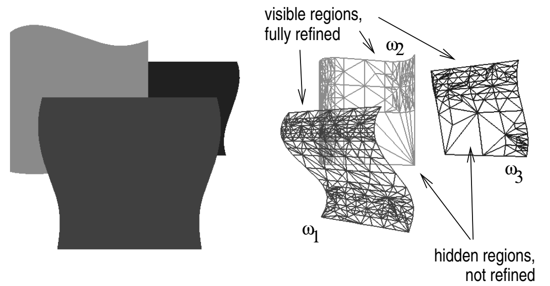

Figure 4: Front to back refinement. The right picture is a snapshot from an additional viewpoint, revealing the unrefined regions

LIGHTING

After the direct visibility is calculated, the lighting is calculated. Each visible vertex emits rays to all light sources. All simplex edges, which have different impact of light, are refined. No refinement is done, if there is equal light impact per edge or the edge properties deny any further. As a consequence, all simplices covering a visible light/shadow border of a light source will refine and produce sharp shadow contours to the viewer. In Figure 5 some light/shadow contours are illustrated.

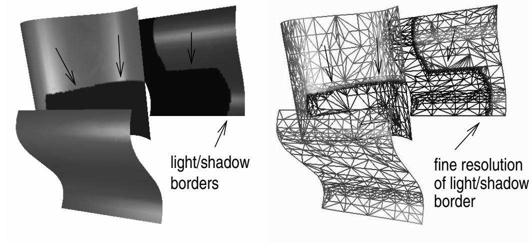

Figure 5: Light/shadow borders. The right picture shows the borders with fine resolution

Surface areas containing specular highlights are recognized by comparing the vertex colors of a simplex. If the set of visible light sources per vertex is equal, but the distance on the color-cube per edge is beyond a user-defined limit, the simplex is further refined. In order to calculate the effects with phong shading like quality we need the surface normals for every simplex edge introduced. This is satisfied within our model. A special treatment is needed, if light rays are blocked by invisible simplices. These indirect visible simplices must be refined first. The refinement must be done regarding to its indirect distance to the viewer.

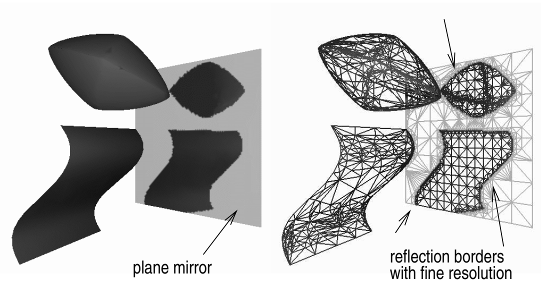

Figure 6: Reflection example. The mirrored patch borders show a fine resolution

REFLECTION

The principle of adaptive refinement can also be applied to calculate reflection and refraction effects. The surface normal per vertex is used, to calculate correct secondary rays per visible vertex. A simplex edge with secondary rays hitting different patches is refined. If all secondary rays of one simplex hit the same patch, the refinement can be continued until every secondary ray hits the same simplex and/or the hit-point colors do not differ too much. Of course, the refinement process is stopped, if the simplex projection size remains under a predefined limit. Figure 6 shows an reflection example. Like the lighting calculation, invisible simplices hit by secondary rays must be refined properly before. In Figure 7 you can see a refraction example. A yellow plain surface is visible through a wavy water-like surface above.

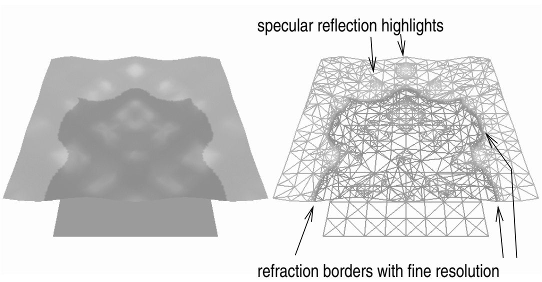

Figure 7: Refraction example. The patch borders show also a fine resolution

RADIOSITY

The principles of adaptive subdivision are well studied within radiosity applications [Campb90]. The subdivision of surfaces into discrete areas must correspond to the areas with different energy transfers (radiance). By the use of ray-tracing, the subdivision process can be controlled [Pauli94], also for bicubic bézier surfaces [Nishi93]. Some of the applied principles are similar to ours. Within our algorithm, we combine the subdivision scheme based on vertex properties with a discrete parametric surface representation. Until now the light geometry is limited to points as ray-tracing, so the calculation of penumbra is not supported. The usability of our subdivision for radiosity calculations is an area of future work.

4 RESULTS

The number of objects, rays and intersection tests are the critical parameters within ray tracing. We evaluated the effort per frame of our algorithm in this regard. The test scenery is build up by multiple instances of cylinders. Each cylinder is represented by an adaptive simplicial complex, consisting of four parametric patches. The objects were placed in a regular cubic 3d-grid. Figure 8 shows an example.

Figure 8: 27 cylindric parametric surfaces with one light source

In Figure 9 the same example with reflection is shown. The error limits were kept fix throughout the test. Only visibility and lighting was calculated. The picture size was $512 \times 512 = 262144$ pixel, lighting is done by one white light. All objects were kept within the viewing frustum by a proper scaling. The calculations times are from a Pentium II 450 MHz Xeon processor linux system.

Obj.	Simpl.	Vert.	Rays	Int.Tests	s
1	3108	1930	1284	234754	<2
8	14190	8528	6528	1521762	6
27	25998	14029	12840	2479382	9
64	31910	16072	16824	2646302	15
125	45512	23426	25247	3459466	26
216	62420	31830	35671	5116677	56
343	72382	41088	43505	5774481	100
512	88108	51811	54925	7551339	175

Table 1: Algorithm effort for increasing scene complexity. (*Int.Tests = Intersection Tests, s = Time in seconds*)

It is difficult to compare this results with standard ray tracing. How many simplices will be needed to produce comparable geometric shapes within ray tracing? However, the results in Table 1 show clearly, that regarding to the total number of rays, our algorithm is better then standard ray tracing[6]. And due to hierarchical simplex structure the number of intersection-tests between rays and simplices is always less then $\sharp$rays$\times\sharp$simplices.

Figure 9: 27 cylindric parametric surfaces with one light source and primary reflections

The discrete representation base and the ray tracing per vertex is a possible source of errors. A coarse object representation may not refine sufficient, because its vertices are hidden, while the whole object is not. It is very important to have a good split up into patches per parametric surface to alleviate this problem. The user-defined error limits of physical space and projection space can also cause additional artefacts. For example, the effects towards genus-preservation and controlled genus-reduction [ElSan97] by this limits is an area of future work.

[6] 262144 pixel will require 262144 rays minimum

An interesting idea is the integration of texturing. The parameter space references u_ν, v_ν per vertex ν can also be used as texture coordinates. Until now, the results of light and secondary rays cause a further refinement of the emitter. If we store the results within a texture instead, the split up by secondary rays can be reduced.

5 CONCLUSIONS

We have presented a modified ray tracing algorithm for parametric surfaces. The concept of adaptive simplicial complexes, composed of injective parametric patches, delivers a uniform interface to the algorithm and a compact scene description. The principle of accuracy on demand throughout the complete algorithm promises low effort per frame. In comparison towards polygon rendering and ray tracing our algorithm is in between. Through the adjustment of error parameters, the quality and effort can be varied continuously. The exploitation of spatial and temporal coherence is an intrinsic algorithm property, also within dynamic scenes.

We do not promise to gain ray tracing like quality with the effort of polygon rendering. Our primary intentions are to find a comprehensive concept for a future graphic subsystems and its interface to the host. As a next step we will investigate the algorithm properties within complex scenes, and the possible application for radiosity. We hope the consequent use of accuracy on demand will lead us to a workload much less than in current algorithms for ray-tracing or radiosity.

ACKNOWLEDGMENTS

We would like to thank Winfried Nöth for many useful discussions and contributions to this project. Also thanks to the anonymous reviewers for their valuable hints.

References

[Berto95] M.Bertolotto, L.De Floriani, P.Marzano, *Pyramidal Simplicial Complexes*, Proceedings of the ACM Symposium on Solid Modeling '95, Salt Lake City, Utah, 1995

[Campb90] A.Campbell,D.Fussell: *Adaptive Mesh Generation for Global Diffuse Illumination*, Computer Graphics, Vol.24, No.4, pp.193-204, 1990

[Catmu75] E.Catmull: *Computer Display of Curved Surfaces*, Proc. IEEE on Computer Graphics, Pattern Recognition and Data Structures, May 1975

[Cohen88] M.Cohen, S.Chen, J.Wallace, D.Greenberg: *A Progressive Refinement Approach to Fast Radiosity Image Generation*, Computer Graphics, Vol. 22, No. 4, 1988

[Cohen96] J.Cohen, A.Varshney, D.Manocha, G.Turk, H.Weber, P.Agarwal, F.Brooks, W.Wright: *Simplification Envelopes*, Computer Graphics Proceedings, Annual Conference Series, pp.119-128, 1996

[DeFlo97] L.De Floriani, P.Magillo, E.Puppo: *Visualizing Parametric Surfaces at Variable Resolution*, Lecture Notes in Computer Science, No. 1311, pp. 308-315, 1997

[ElSan97] J.El-Sana, A.Varshney: *Controlled Simplification of Genus for Polygonal Models*, Proceedings of the IEEE Visualization '97, Oct. 19 - Oct. 24, to appear, 1997

[Encar88] J.Encarnacão, W. Strasser: *Computer Graphics*, Oldenburg Verlag, third edition, 1988

[Foley90] J.Foley, A.Dam, S.Feiner, J.Hughes: *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Company, second edition, 1990

[Funkh93] T.Funkhouser, C.Séquin: *Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments*, Computer Graphics Proceedings, Annual Conference Series, pp. 247-254, 1993

[Heckb84] P.Heckbert, P.Hanrahan: *Beam Tracing Polygonal Objects*, Computer Graphics Proceedings, Annual Conference Series, pp.119-127, 1984

[Hoppe96] H.Hoppe: *Progressive Meshes*, Computer Graphics Proceedings, Annual Conference Series, pp.99-108, 1996

[Jaeni94] K.Känich: *Topologie*, Springer-Verlag, 4. Auflage, 1994

[Klein96] R.Klein, G.Liebich, W.Strasser: *Mesh Reduction with Error Control*, Visualization 96, ACM, pp.311-318, November 1996

[Lane80] J.Lane, R.Riesenfeld: *A Theoretical Development for the Computer Generation of Piecewise Polynomial Surfaces*, IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 35-46, Jan 1980

[Luebk97] D.Luebke, C.Erikson: *View-Dependent Simplification Of Arbitrary Polygonal Environments*, Computer Graphics Proceedings, Annual Conference Series, pp. 199-208, 1997

[Nishi93] T.Nishita, E.Nakamae: *A New Radiosity Approach Using Area Sampling for Parametric Patches*, Computer Graphics Forum (Eurographics '93), pp. C385-C398, 1993

[Pauli94] M.Paulin, J.Jessel: *Adaptive mesh generation for progressive radiosity: A ray-tracing based algorithm*, Computer Graphics Forum (Eurographics '94), pp. C421-C432, 1994

[Popov97] J.Popović, H.Hoppe: *Progressive Simplicial Complexes*, Computer Graphics Proceedings, Annual Conference Series, pp. 217-224, 1997

[Rockw89] A.Rockwood, K.Heaton, T.Davis: *Real-Time Rendering of Trimmed Surfaces*, Computer Graphics, Volume 23, Number 3, pp. 107-116, July 1989,

[Shiny87] M.Shinya, T.Takahashi, S.Naito: *Principles And Applications of Pencil Tracing*, Computer Graphics, Volume 21, Number 4, pp. 45-54, July 1987

[Synde87] J.Synder, A.Barr: *Ray Tracing Complex Models Containing Surface Tessellations*, Computer Graphics, Volume 21, Number 4, pp. 119-128, July 1987

[Toth85] D.Toth: *On Ray Tracing Parametric Surfaces* Computer Graphics, Vol. '9, No 3, pp. 171-179, 1985

[Woodw89] C.Woodward: *Theory and Practice of Geometric Modeling: Ray Tracing Parametric Surfaces by Subdivision in Viewing Plane*, Springer-Verlag, pp. 273-1.5, 1989