Real-time rendering of complex surfaces defined by atlas of discoids

Benoît Piranda Sylvain Magdelaine Didier Arquès
SISAR Team
University of Marne La Vallée
5, boulevard Descartes, Champs sur Marne
77454, Marne La Vallée, France
piranda@univ-mlv.fr ewik@free.fr arques@univ-mlv.fr

ABSTRACT

This paper expounds a new method of complex surfaces rendering permitting to visualize atlas of discoids in real-time. The atlas of discoids allow to define surfaces very easily without the topological constraints we could get with the classical methods such as polygons meshes ones. Our basic model is completed by many algorithms permitting either to reconstruct implicit surfaces or to calculate the global illumination using radiosity algorithm. But up to now, no fast viewing method has been proposed for this modeling algorithm.
We present here a real time rendering algorithm which exploits the recent extensions of OpenGL library and the possibilities offered by GPU like the shaders programs to perform non standard fragments mixing operations.

Keywords
Surface reconstruction, real-time visualization, atlas of discoids.

1. INTRODUCTION

Computer graphics techniques propose a large choice of solutions for surface modeling. The most widely used method consists in defining any surface by a polygonal mesh [Fol90]. Even if a classical structure like the winged-edge data structure [Sil94] allows to represent efficiently such a mesh, many drawbacks exist. In a rendering process, visual artifacts due to preponderant directions (the edges of the polygons) appear. Manipulation such as patch subdivisions become complex (see for instance [Man88] for a complete overview about solid modeling) because topological constraints between neighboring patches have to be maintained. We can also point out that similar topological problems appear in the surfaces reconstruction process of from a set of points [Boi84].

An interesting alternative consists in using Spline [Bar89] or Nurbs [Roc89] surfaces which model any surface by a collection of piecewise-polynomial patches instead of previous planar ones. More complex surfaces can be modeled but the main drawbacks concern continuity constraints in the junction points [Wat93] and rendering computation time generated by ray-tracing algorithm.

Different interesting works aim at combining the previous approaches. Szeliski and Tonnesen [Sze92] propose to use disks or particles to represent the mesh describing an implicit surface. Oriented particles interact each other according to repulsion and attraction forces to automatically treat modifications of the surface (split, join, extend). Indeed, Witkin and Heckbert [Wit94] use oriented particles to sample regularly an implicit surface.

The recent point-base rendering model allows to visualize a surface described by a set of points. Splats defined in [Zwi01] are commonly represented by disks centered on these points, and that may overlap each others. These splats are projected onto the screen and filtered by a Gaussian kernel in order to represent a continuous textured and illuminated surface.

Other recent works [Arq00a] present another method using elements of surfaces (discoids) that can overlap each others. So, defining such a surface with a set of disk-like patches imposes few constraints. For example, it avoids completely topological constraints associated to a classical mesh. On the contrary, overlapping areas are welcomed. We just have to place surface elements in the area we want the surface to exist, and we choose the geometry and the size of each discoid in order to obtain a "total covering" of the surface (cf. figure 1).

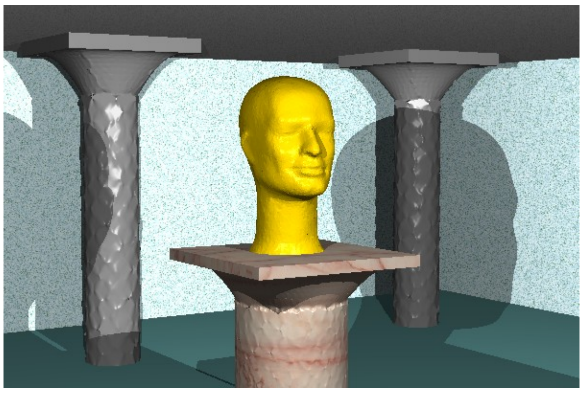

Figure 1: scene completely defined by atlas of discoids.

In [Arq00b], this geometrical model is completed by a rendering algorithm based on the radiosity global illumination model. However, in the two former papers, the visualization of surfaces described by atlas of discoids suffers the drawbacks of the ray-tracing algorithm, a calculation time that doesn't allow real-time utilization.

The following image shows a human face made of only 19 more general discoids defined by overlapping polynomial surfaces.

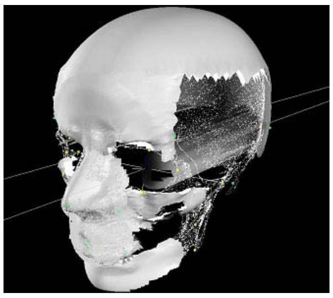

Figure 2 : surface defined by 19 discoids.

Recent new capacities of the OpenGL library in programming the Z-Buffer algorithm allow to intervene in the heart of the rendering process developing short programs : the shaders. One of the categories of these short programs, called "fragment shaders", allows to modify the algorithm that selects the visible polygon in order to determine the color of the filled pixel.

In this paper, we propose a new method permitting to visualize surfaces reconstructed from an atlas of simple surfaces that overlap each other in real-time using recent capacities of the OpenGL library in programming the Z-Buffer algorithm. We first present the atlas of discoids model, then we show how use shaders programs to calculate the melting of overlapping discoids. Then, we present some results on images.

2. ATLAS OF DISCOIDS AND Z-BUFFER

The atlas of discoid

On a given surface *S*, the atlas system allows to reconstruct any interest function *F*, for instance a temperature or luminance function from an atlas of discoids. We define an atlas of discoids as a set of *N* disks-like patches $\{D_i, i=1\ldots N\}$ that cover entirely but approximately the surface *S* without taking into account overlapping problems. (cf. figure 3).

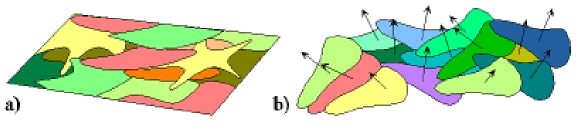

Figure 3 : example of planar and complex surfaces defined by an atlas of discoids.

Only for understanding reasons and notation simplifications, we just consider in this section the case of a planar surface *S*. Any point *M* of *S* is covered by a subset of discoids as shown in figure 4. Then the value of the interest function *F* in *M* is defined by :

$$F(M) = \sum_{i / M \in D_i} \alpha_i(M) F_i(M) \qquad (1)$$

where :

$F_i(M)$ is the local interest function defined for each point of the disk D_i covering *M*;

α_i is a merging operator, defined and positive on each disk D_i, and which verifies $\sum_{i / M \in D_i} \alpha_i(M) = 1$.

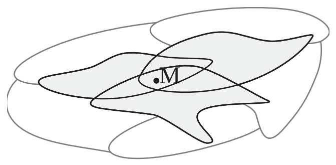

Figure 4 : only gray disks are used to define the interest function in M.

The choice of the function α_i for each disk D_i is relatively free. It can be seen as a decomposition of the interest function in a function base with a local geometrical support: the discoid. An interesting method consists in choosing any set of positive functions β_i and to define α_i by:

$$\alpha_i(M) = \frac{\beta_i(M)}{\sum_{j / M \in D_j} \beta_j(M)} \qquad (2)$$

For example, the β_i functions may depend on the distance from *M* to the center of the discoid. By choosing a function β_i which varies continuously from 1 in the center to 0 in the border of each disk, it is easy to verify that the merging operator runs as a smoothing operator only in the overlapping area.

In this paper, we propose to define the β_i function over the discoid with a gray level texture for each discoid (cf. figure 5). Black points are null values of β_i and the white ones correspond to $\beta_i = 1$.

Just notice that null values of the β_i define the boundaries of the discoid.

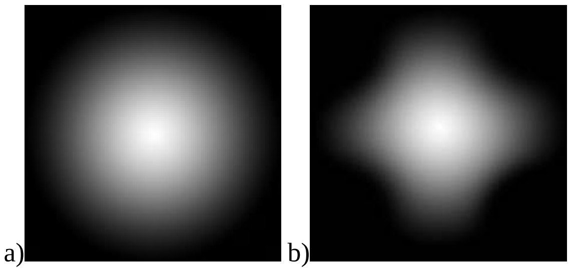

Figure 5: two shapes of discoids defined by a texture : a disk (a) and a star (b)

Z-Buffer and fragment

First, let's remind the well-known Z-buffer algorithm, and its implementation in the OpenGL library. The Z-Buffer algorithm consists in painting every polygon of the scene. For each pixel filled by the polygon, the process emits a "fragment" that contains the geometrical information and illumination coefficients of the polygon. One of these fragments is selected to be used to color the corresponding pixel in the color buffer.

Fragment shaders programs allow to develop our own treatments applied to each fragment. For example, we have developed a short fragment program that calculates the illumination applying Phong [Pho75] illumination model in the level of the fragment. By using this fragment program instead of classical OpenGL lighting program, we obtain a smoother surface aspect as shown on figure 6.

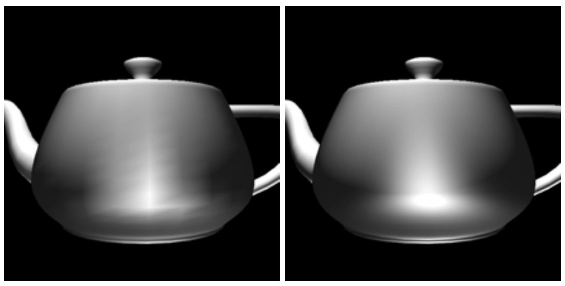

Figure 6: the same model rendered using OpenGL method (left) and with fragment program (right).

3. ADAPTATIONS FOR REAL-TIME VISUALISATION

Geometrical considerations

If we consider now the general case, the disks do not exactly cover the surface *S*. We have to associate to the point *M* and for each disk D_i a point M_i. Equation (1) becomes:

$$F(M) = \sum_{i / M \in D_i} \alpha_i(M_i) F_i(M_i) \qquad (3)$$

Using the Z-Buffer algorithm to render the surface, we mix the points M_i projected on the point M of the surface. If we consider a point of view *O* and the direction of viewing $\vec{u}$, the points M_i correspond to the intersection point between the ray $[O\vec{u})$ and the discoid D_i as shown in the figure 7.

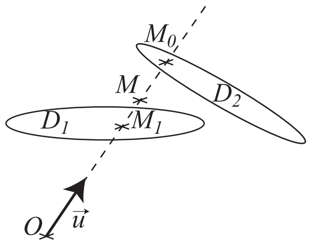

Figure 7: association of points *M_i* of discoids to point *M* of the surface.

Then during the Z-Buffer process, the pixel on with the point M is projected receives a fragment for each discoid that covers *M*. Then, knowing the value β_i (memorized in the gray level texture) for the *n* discoids M_i projected in M, we deduce from equation (3):

$$F(M) = \frac{\sum_{i=1}^{n} \beta_i(M_i) F(M_i)}{\sum_{i=1}^{n} \beta_i(M_i)} \qquad (4)$$

In the Z-Buffer algorithm, the denominator part of the equation (5) is only known at the end of the process, ie when all fragments are arrived. So, in order to calculate the interest function of the surface as soon as the fragments arrive, we express the equation (4) under a new incremental formulation:

$$\begin{cases} F^{(0)}(M) = 0 \\ F^{(t)}(M) = \dfrac{F^{(t-1)}(M) \sum_{i=1}^{t-1} \beta_i(M_i) + \beta_t(M_t) F^{(t)}(M)}{\sum_{i=1}^{t} \beta_i(M_i)} \end{cases} \qquad (5)$$

Where *t* is the order of the fragments' arrival.

Level of details

More precisely, a problem of level of details appears. Considering all discoids that produce a fragment for the same pixel, we have to melt only discoids that effectively cover the same point of the surface. We don't have to mix far away discoids as shown in figure 8 and presented in a practical case in the image of figure 12 (where the two skulls are faraway).

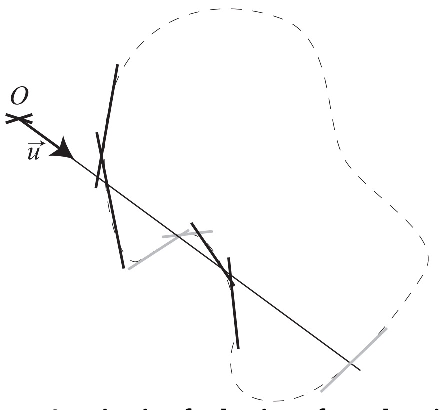

Figure 8: criteria of selection of overlapping discoids

Two criteria are evaluated: the orientation of discoids and the distance between discoids along the vision axis. We use OpenGL culling test to delete fragments corresponding to the bad-oriented discoids (in gray in figure 8) and we use an adapted depth test to select which fragments should be mixed. We melt only the discoids that are very near to the memorize surface: when the depth Z of a new fragment is very different to the memorized depth Z_{mem}, we apply the classical algorithm of Z-Buffer.

More precisely, we first calculate the depth map of the scene without melting discoids and save it in a texture (called "depth" in the following code). Then during the other steps of the program, we only treat the discoids placed near this visible surface. The threshold is a level of details parameter, it must be chosen much smaller than the dimension of the discoids.

```
// texture memorizing the depth buffer
uniform sampler2D depth;
// texture of beta values
uniform sampler2D discoid;
// threshold value
uniform float t;

void main (void)
{ float beta = texture2D(discoid,
                  vec2(gl_TexCoord[0])).a;
  float dpt=texture2D(depth,
               vec2(gl_FragCoord/size)).x;
// depth buffer
  if(beta==0.0 || gl_FragCoord.z > dpt+t)
  { discard;
  }
}
```

For example, in figure 9 the same scene is drawn with a excessive (left) and good (right) value of threshold. We can observe some artefacts due to the mixing of too far away discoids.

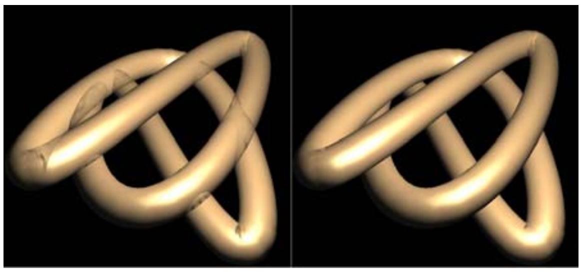

Figure 9: a scene rendered with two different values of threshold

Illumination model

The illumination model used in our implementation is based on Phong illumination model [Pho75] applied on each fragment. The luminance L is calculated using the following expression:

$$L = K_a + K_d \cos(\alpha) + K_s \cos^N(\gamma) \qquad (6)$$

Where K_a is the ambient term, K_d the Lambertian diffusion coefficient, K_s the specular reflexion coefficient, and N the shininess Phong coefficient.

The first developed solution doesn't use the MRT (Multi Render Target) library, so we have to write illumination parameters in only 4 bytes (RGBA) per fragment. So we use two times the equation (5) with $\cos(\alpha)$ and $\cos^N(\gamma)$ as interest function. After incremental calculation, the result is placed in the RVBA memory according to the following table. The two last bytes are reserved to memorize the denominator part of equation (5).

R	G	B	A
$\sum_{i=1}^{t} \beta_i(M_i)\cos\alpha_i$	$\sum_{i=1}^{t} \beta_i(M_i)\cos^N\gamma_i$	$\sum_{i=1}^{t} \beta_i(M_i)$	

Table 1. illumination data of a fragment

It is also possible to directly calculate the $\sum_{i=1}^{t} \beta_i(M_i)$ term using the blending capacities of the OpenGL library. Each fragment sends its β_i value coded in two bytes of the fragment color (first 4 bits in A and last 4 bits in B) and then the blending process computes the sum in the color buffer.

```
// texture containing beta values
uniform sampler2D discoid;
void main (void)
{ // beta value on the fragment
  float beta = texture2D(discoid,
vec2(gl_TexCoord[0])).a;
  float coef = beta*256.;
  gl_FragColor.b = floor(coef/16.);
  coef-=gl_FragColor.b*16.;
  gl_FragColor.b/=256.;
  gl_FragColor.a = coef/256.;
}
```

We just have to copy this buffer in a texture in order to use it in the next steps of the program. The value of the $\sum_{i=1}^{t} \beta_i(M_i)$ term is finally obtained combining the B and A components of the texture : Bx16 + A.

```
// texture containing encoded sumBeta values
uniform sampler2D sumBeta;

void main (void)
{ // sBcode.ba : encoded values of sumBeta for the
current fragment
  vec4 sBcode = texture2D(sumBeta, vec2(gl_FragCoord
/ size));
  float sumBeta = sBcode.b*16. + sBcode.a;
}
```

The following figure shows a sphere defined by 20 discoids, with the beta function presented on figure 5a. The first image (on left) shows the placement of the patches carrying the discoids. The image on the right is obtained by melting the previous discoids using our algorithm and applying a per fragment Phong lighting program.

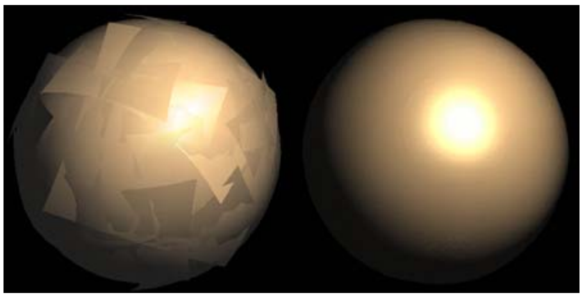
Figure 10: A sphere covered by 20 curved discoids

4. RESULTS

The two following examples presented below show surfaces described by atlas of discoids rendered by our program. The frame rate of these two examples is about 20 images per second on a NVidia Geforce 6 6800 GT Graphic card.

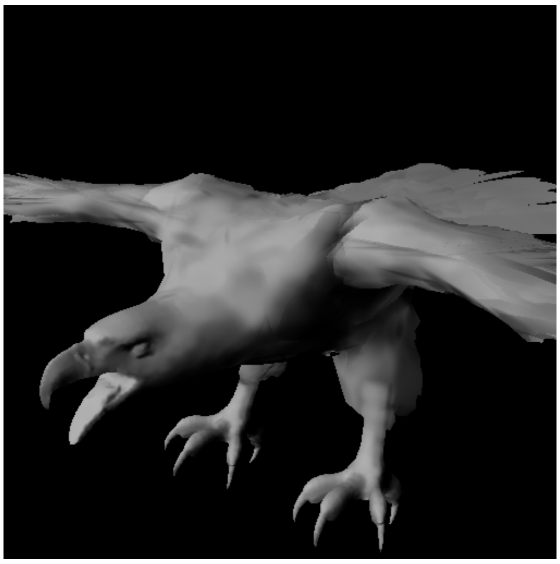
Figure 11: Eagle model defined by 8800 discoids

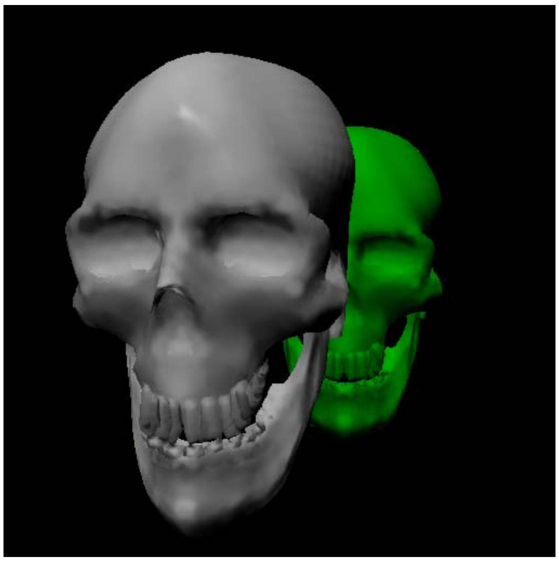
Figure 12: Two skulls by 7176 planed discoids each.

5. CONCLUSION AND FUTURE WORKS

In this paper we propose a new algorithm permitting to visualize in real-time, surfaces defined by an atlas of discoids. This approach reduces topological constraints by simply describing any object by an atlas of discoids.

Such method combined to a real-time visualization process is the first step to the development of a surface modeler that creates continuous surface from natural drawing of shapes.

With the last new capacities proposed by OpenGL 2.0, and more precisely MRT (Multiple Render Target), it is possible to memorize many interest functions for each discoid, written in different buffers. Moreover, the using of floating point textures allows better precision of calculation. These improvements will permit to construct surfaces associated to more illumination parameters and better speed of calculation.

6. REFERENCES

[Arq00b] D. Arques, S. Michelin and B. Piranda, Overlapping radiosity: using a new function base with local disk support, WSCG'2000, vol. 3, 2000, pp. 236-243.

[Arq00a] D. Arquès, S. Michelin, B. Piranda. Modelisation of Implicit Surfaces Driven by an Atlas of Discoids, GraphiCon'2000, 2000.

[Bar89] R.H. Bartels and C.J. Beatty, A technique for the direct manipulation of spline curves, Graphics Interface’89, 1989, pp. 33-39.

[Boi84] J.D. Boissonnat, Geometric structure for three dimensional shape representation, ACM

Transactions on Graphics, vol. 3(4), 1984, pp. 266-286.

[Fol90] J. Foley, A. Van Dam, S. Feiner and J. Hugues, Computer graphics: principles and practice, 2nd edition, Addison Wesley, 1990.

[Man88] M. Mäntylä, An introduction to solid modeling, Computer Science Press, 1988.

[Pho75] B. T. Phong. Illumination for computer generated pictures. Communication of the ACM, Vol.18, n°6, 311-317, 1975.

[Roc89] A. Rockwood, K. Heaton and T. Davis, Real-Time Rendering of Trimmed Surfaces, SIGGRAPH '89 Proceedings, vol. 23(3), 1989.

[Sil94] F.X. Sillion and C. Puech, Radiosity and global illumination, Morgan Kaufmann publisher, 1994.

[Sze92] R. Szelisky and D. Tonnesen, Surface Modelling with Oriented Particle Systems, Computer Graphics, vol. 26(2), 1992, pp.185-194.

[Wat93] A. Watt, 3D Computer Graphics, 2nd edition, Addison Wesley, 1993

[Wit94] A.P. Witkin and P.S. Heckbert, Using Particles to Sample and Control Implicit Surfaces, SIGGRAPH '94 Proceedings, 1994, pp. 269–277.

[Zwi01] M. Zwicker, H. Pfister, J. van Baar and M. Gross, Surface splatting, In proceedings of ACM SIGGRAPH 2001 (2001), pp. 371-378.