Interactive Solid Texturing using Point-based Multiresolution Representations

Patrick Reuter
LaBRI - CNRS - INRIA
University Bordeaux 1
France
preuter@labri.fr

Benjamin Schmitt
LaBRI - CNRS - INRIA
University Bordeaux 1
France
schmitt@labri.fr

Christophe Schlick
LaBRI - CNRS - INRIA
University Bordeaux 1
France
schlick@labri.fr

Alexander Pasko
Dept. Digital Media
Hosei University
Japan
pasko@mail.com

ABSTRACT

This paper presents an interactive environment for texturing surfaces of arbitrary 3D objects. By uniquely using solid textures and applying them to the surface, we do not require an explicit parameterisation in texture space. Various solid textures can be combined by building a constructive texturing tree of space partitions to define the photometric attributes at each location of the object. Though solid texturing using constructive textures is very powerful, mainly because of its generality, it is quite difficult to use in practice as it is not well suited to interactive tools. To overcome this limitation, we use a multiresolution point-based representation ensuring that texture evaluation and rendering maintains a given frame rate. Our tool is realized as a plugin for the *Pointshop3D* system. The main advantage of our texturing approach compared to Pointshop3D is that point-based rendering is only used during the interactive texturing step. We always keep a feedback to the initial geometric representation of the object (polygonal mesh, parametric or implicit surface, voxel arrays, or whatever) which means that the final textured object can be easily exported to standard graphics software that cannot directly handle discrete surface points (e.g. CAD systems, photorealistic rendering engines).

Keywords: solid texturing, point-based representations, constructive texturing

1 INTRODUCTION

The typical process used in the computer graphics industry to create a realistic 3D object has not much changed during the last twenty years. There are three principal steps involved in this process: first, define the shape of the object by creating a geometric representation, second define its visual appearance by creating a set of surface or volume textures, and third, apply the textures all over the geometry to get a nice-looking final object. A huge number of techniques have been proposed to achieve the first step, a large number of techniques have also been proposed to achieve the second one, but only few methods have been proposed to easily and robustly link the texture and the geometry.

Journal of WSCG, Vol.12, No.1-3., ISSN 1213-6972
WSCG'2004, February 2-6, 2004, Plzen, Czech Republic

Except for some simple shapes and/or simple textures, this third step is still a painful trial-and-error task to do for the graphics designer, especially when the texture has to behave accordingly to the geometry changes in an animated sequence.

In this paper, we propose a new idea that links the texture and the geometry of arbitrary objects during an interactive and intuitive process. This process combines two approaches previously developed in computer graphics: *constructive texturing* and *point-based rendering*. The first approach offers a general framework for texturing objects of arbitrary nature (i.e. polygonal meshes, voxel arrays, parametric or implicit surfaces, and others), while the second one offers a flexible way for rendering the surfaces of such objects. The combination of both approaches allows the development of an intuitive tool for applying textures to geometric shapes with interactive feedback. This interactivity is guaranteed, whatever the complexity of the geometry and the texture, by using a multiresolution representation of discrete surface points extracted from the object, both for rendering and for texture evaluation.

The remainder of the paper is organised as follows: Section 2 recalls some previous work on which we have built our

new approach, Section 3 presents our technique in detail, Section 4 discusses some of our experimental results, and finally, Section 5 concludes and proposes some directions to future work.

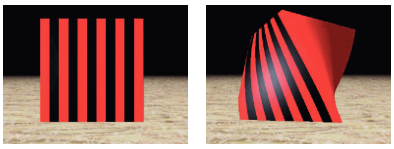
2 PREVIOUS WORK

2.1 Constructive texturing

In [Schmi01], a general approach for texturing objects of arbitrary nature, called *constructive texturing*, is introduced.

This technique consists in defining for a given object G a partition $\Pi = \bigcup_i \Pi_i$, where in each subset Π_i of the partition, a different set of photometric attributes is defined (ambient, diffuse and specular colours, reflectance and transmittance coefficients, etc.). For each point of the object, one has to be able to answer the following question: "Is this point inside or outside a given subset Π_i?" In the affirmative case, the photometric attributes corresponding to Π_i are applied.

One powerful solution for point membership classification is to use the function representation (FRep, for short) model [Pasko95]. With this model, each space partition Π_i is defined by a function F_i, corresponding to an attribute A_i. The only requirement of the FRep model is to define F_i as a C^0 real-valued function, with negative values outside the partition, positive values inside the partition, and zero values on its boundary. The defining functions F_i are usually built using a constructive approach, resulting in a tree structure, with primitives at the leaves and operations at the nodes. This tree can be seen as an extension of the classical CSG tree where the user can easily provide his own set of primitives and operations.



(a) textured cube (b) twisted geometry and texture

Figure 1: The texture follows after applying a twist operator.

The main advantage of constructive texturing is its generality: it can be applied to polygonal meshes, BRep representation, parametric and implicit surfaces, voxel arrays, and others. Moreover, the texture behaves accordingly to geometry changes in animated sequences without creating distortions as illustrated in Figure 1. On the other hand, the major drawback of this approach is that the creation of the space partitions is usually complex and rather painful. This is due both to the constructive approach inherent to method, and to the lack of interactive tools. Consequently, it is difficult for a non-expert user to generate the nodes of the partition at the desired location of the object surface.

2.2 Point-based rendering

Although the first use of points as rendering primitive can be attributed to Levoy and Whitted in 1985 [Levoy85] and to Szeliski and Tonnesen in 1992 [Szeli92], *point-based rendering* has attracted growing interest since 1998 starting by the works of Grossman and Dally [Gross98]. Further developments by Rusinkiewicz and Levoy [Rusin00] and Pfister et al. [Pfist00] have generated dozens of papers recently. This is due to the fact that modern graphics hardware has made it possible to treat and render an order of magnitude higher amount of points. A similar trend in the modelling field, that could be called *point-based modelling*, can be detected in a couple of recent papers [Pauly03, Reute03, Turk02]. The main idea that links all these papers together is the use of discrete surface points without explicit connectivity instead of polygonal meshes. As the connectivity has no longer to be managed, some common operations, such as multiresolution (level-of-detail) generation, geometrical deformations, or topology modifications, are much easier to implement with objects described by discrete surface points compared to polygonal meshes.

2.3 Interactive surface painting and texturing

In the literature, several environments can be found, which can be used to interactively paint or texture the surface of an object. Surface painting environments are usually based on a metaphor of the real painting approach, where one uses a symbolic pencil or a brush to paint the object, the colour being applied is then evaluated according to a brush function resulting in a realistic effect [Hanra90, Agraw95]. Surface texturing environments require first to specify a location and a direction on the surface, and second, to apply an existing 2D image. The latter step requires a local parameterisation of the surface which is often a non-trivial task. Furthermore, one has to think how to stitch the 2D images together without distortion. Several solutions exist, Praun et al. [Praun00] show how to repeatedly map a small texture on a polygonal mesh using local overlapping parameterisations, and Turk [Turk99] and Wei and Levoy [Wei01] show how to synthesise textures on polygonal meshes.

In almost every existing surface painting and texturing environment, a polygonal mesh or a parametric patch is required. When the object is defined by polygons, no additional step is needed. But if it is defined using another representation scheme, a special preprocessing is required. For instance, Pedersen [Peder95] proposes a general solution when the object is defined as an implicit surface, where a local parameterisation is defined by estimating geodesics on the surface. Witkin and Heckbert's particle system [Witki94] can also be used to establish a parameterization for texture mapping on implicit surfaces [Zonen98].

One solution to avoid parameterisation when texturing surfaces of arbitrary objects is the use of octree textures developed recently [Benso02, DeBry02], where only the subsets of volume textures actually intersecting the surface are stored efficiently in an octree. However, the resulting texture is discrete and has a fixed resolution limit determined by the depth of the octree. Moreover, an octree has to be stored for every attribute, resulting in a significant storage

overhead. Nevertheless, octree textures can be easily integrated in our software environment thanks to the constructive texturing approach.

Another solution that does not require explicit parameterisation is used in the Pointshop3D environment [Zwick02]. This innovating approach proposes to paint the surface of an object which is defined as a cloud of points. By applying a texture (2D texture, uniform colour, or other), the corresponding surface points are coloured. The interactive process of painting is intuitive, and good results are achieved. However, as it is totally based on discrete surface points, this approach suffers from a severe drawback: once the surface is textured, the resulting object can hardly be exported to standard graphics software that cannot handle discrete surface points (e.g. CAD systems, photorealistic rendering engines).

3 OUR APPROACH

3.1 Overview

The main advantage of solid texturing is that it can be applied on surfaces of arbitrary 3D objects without requiring a parameterisation. On the other hand, creating complex solid textures using constructive texturing and applying them at the desired location of the surface is difficult for a non-expert user. The lack of interactive and intuitive tools makes this approach painful because it is hard to generate the nodes of the texture partition according to the shape of the object. This paper proposes a solution that solves this problem: its basic idea is to let the user define the space partition by interactively selecting points on the surface.

Interactivity is guaranteed thanks to the dual nature of the multiresolution scheme being applied. First, as usual, the multiresolution representation is used for rendering to maintain a constant frame rate. Thanks to the adaptive and flexible multiresolution representation, areas of interest can be rendered in a higher resolution than the other areas of an object surface without causing connectivity problems. Second, in addition to rendering, we also use the multiresolution approach in the texturing step: at low resolution, the user can paint large parts of the object, and when the resolution is increased, finer details can be painted. Moreover, the photometric attributes can also be determined in a multiresolution manner, thus, when texturing an object surface, visual results are obtained rapidly in a coarser resolution before refining them in a background process.

The procedure we have defined can be divided into 10 steps, as illustrated in Figure 2. In a preprocessing step, discrete surface points are extracted from a given object of arbitrary type (step 1), and a multiresolution representation of the cloud of discrete surface points is set up (step 2). After this, the object is visualised in the adapted level-of-detail (step 3). Then, the object surface can be textured by the user (steps 4-7), the photometric attributes of the surface points are updated (step 8), and visualised (back to step 3). Finally, when the user has finished texturing, the texture can be exported (step 9) and used during postprocessing (step 10). Details of these steps are given in the following subsections.

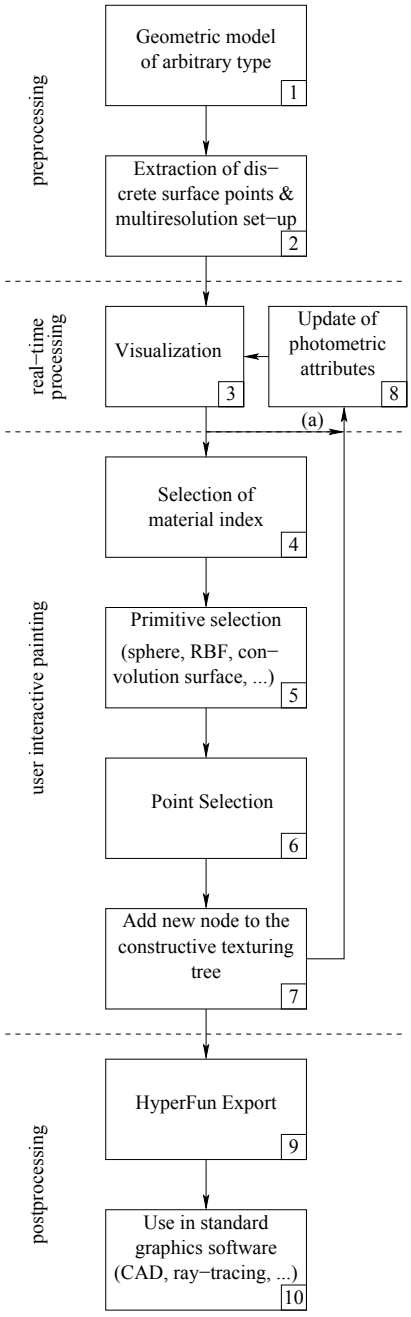


Figure 2: The different steps involved in our interactive texturing process.

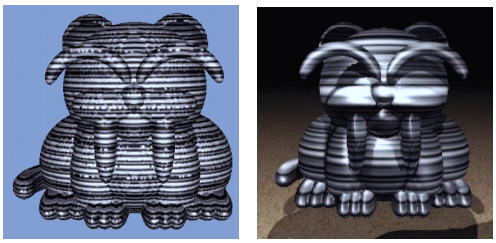
3.2 Preprocessing

Before texturing an object using a point-based rendering technique, one needs to define a cloud of discrete points on the object surface (step 1). The complexity of this preprocessing stage heavily depends on the object nature. In the case of polygonal or parametric objects, the extraction is done in a direct manner as these representations explicitly define the boundary of the object. In the case of other representations, such as FRep, voxel arrays, or scanned data, one

needs to extract the corresponding isosurfaces. A large number of techniques exist for this task, such as polygonisation, particle systems [Witki94], or ray-tracing applied to an object. Moreover, some efficient resampling techniques have been proposed in order to augment or lessen the number of extracted surface points [Alexa01].

Once the cloud of discrete surface points is extracted, the multiresolution representation is set up (step 2). This is done not only for level-of-detail rendering, but also for progressive evaluation of the constructive texturing tree. We use a hierarchy of bounding spheres stored in a binary tree in spirit of QSplat [Rusin00] which is built up in a top-down recursive manner during preprocessing. This is done by splitting the set of surface points along the longest axis of its bounding box, recursively computing two subtrees, and finding the bounding sphere enclosing two children spheres. In each node, we store the radius of the bounding sphere and the surface point lying closest to the barycenter of all surface points in the bounding sphere. Note that the radius of the bounding spheres at the leaf nodes is determined directly from the sampling grid used for extracting the surface points.

Defining solid texture coordinates by discrete surface point locations is prone to aliasing artifacts when high-frequency textures are used, even at the highest resolution of the hierarchy. This is also true for the surface textures natively defined in the Pointshop3D environment [Zwick02], but in contrast to Pointshop3D, discrete surface points are only used in an intermediate step to previsualise the textured surface. These aliasing artifacts will not occur during postprocessing as illustrated in Figure 3, where a high-frequency perlin noise is applied to an FRep tiger object.



(a) Previsualisation (b) Final ray-traced image

Figure 3: Aliasing artifacts do not occur in postprocessing.

Figure 4: A set of established space partitions shown on the dinosaur statue.

3.3 Real-time processing

In order to ensure real-time processing, the evaluation of the photometric attributes of the surface points as well as the rendering itself has to be done in a given time. To meet this requirement, the adapted level in the multiresolution hierarchy, where the photometric attributes of all nodes can be evaluated in the given time, is determined. This might not be the highest level of the hierarchy, as the cost of determining the attributes for a surface point increases with the complexity of the solid texture represented by the constructive texturing tree. Then, the photometric attributes of all further nodes and surface points of the multiresolution hierarchy are determined (step 8) in a background process as indicated by (a) in the algorithm.

In addition to choosing the adapted multiresolution level with respect to the evaluation cost of the attribute tree, the number of surface points which can be rendered by the graphics hardware has also to be taken into account for efficient visualisation (step 3). This cost might be rather small when the projection of the bounding sphere of the surface point falls on a single screen pixel. Even when the projection of the bounding sphere falls on several screen pixels, the cost might stay small using some hardware-accelerated splatting technique [Rusin00], but increases when a high-quality splat is drawn [Zwick01]. However, even high-quality splats can now be drawn with hardware acceleration [Ren02] using modern graphics hardware [Lindh01].

3.4 Texturing: user interaction

At this stage of the algorithm, the user textures the object surface. This is done by choosing a material index for the texture to be used (step 4), defining a space partition (steps 5 and 6), and adding the space partition as a new node in the constructive texturing tree (step 7). We will detail these steps in the following.

Selection of material index (step 4) First, the user selects a material index for a solid texture that will be applied in the space partition. Any kind of solid texture can be used, varying from procedural textures, volumetric textures, simple materials, and others.

Primitive Selection (step 5) Depending on the shape of the space partition the user wishes to obtain, different tools are available. By tool, we mean any primitive that can be defined in the FRep model. The simplest tools to define a space partition are the sphere and the block. By using convolution surfaces for the space partition, the user can texture the object with a brush tool of a any size. With more complex primitives, the user has even more control over the space partition to define. For instance, by using variational implicit surfaces [Savch95, Turk99] defined by radial basis functions (RBF), the user can define a volume including all selected points. As an example, a textured object and its corresponding space partitions of a dinosaur sculpture can be seen in Figure 4.

Point Selection (step 6) In this step, the user selects the surface points. These surface points are the parameters of the FRep primitive defining the space partition. For example, by using the RBF primitive, a

space partition interpolating all the surface points is determined. When the chosen tool is a brush, the selected surface points define the skeleton of the convolution surface for the space partition. When a sphere is chosen, the selected surface point defines the center, and the radius can be specified interactively.

New node in the constructive texturing tree (step 7)
Once the new space partition has been created (i.e. a new primitive), it is added to the current constructive texturing tree. This is done automatically while using the set-theorethic union operation, but any other operation available in the FRep model can be used (including other set-theoretic operations such as intersection, or blending union). In the case of overlapping partitions, set-theoretic operations need to be defined by the user. Indeed, if one considers a union operation of, for instance, two overlapping blocks, the resulting geometry is well defined. But if one considers the union of attributes, the result needs to be specified. In the case of a red and a green block, the color of the intersection of the blocks can be either red, or green, or yellow, or any other color; it corresponds to different operations on attributes, namely priority given to an attribute, a min/max function, or a user defined operation. By default, we give priority to the last added primitive.

When the user has painted on the surface by choosing the desired texture and points to define the space partition, the photometric attributes of the surface points are updated (step 8) and visualised (back to step 3). Then, the user can continue to add further space partitions and to associate photometric attributes with them. Although the space partitions can be very complex, interactivity is achieved because the evaluation of the solid texture represented by the constructive texturing tree is processed only for the discrete surface points used for visualisation, at the adapted level-of-detail.

3.5 Postprocessing

Once the solid texture defined by the constructive texture is created, it can be saved, and exported using the extended version of HyperFun [Adzhi99] (step 9), which is a special high-level language that supports all the main notions of FRep modelling and has been recently extended to support the constructive texturing concept. A set of plug-ins has been developed such that several existing software tools support objects described by HyperFun, such as Maya [Maya], PovRay [PovRa], and other (step 10). Export to polygonal representations, such as VRML, are also supported.

4 RESULTS

4.1 Overview

We implemented our tool as a plugin for Pointshop3D [Zwick02]. A screenshot of our plugin can be seen in Figure 5, where the user is selecting the tool to define a space partition. Besides the rendering modes provided by Pointshop3D, we implemented our own rendering modes to manage the multiresolution representation. All timings given in this section were measured on a 1.7 Ghz Pentium PC with 512 MB RAM.

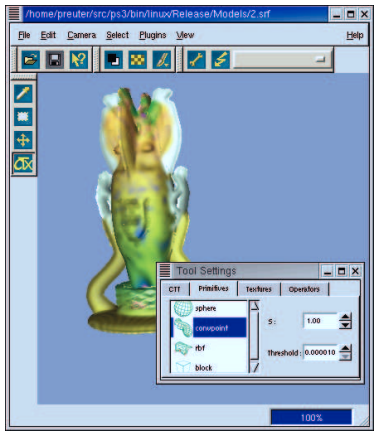


Figure 5: A screenshot of our plugin.

4.2 Preprocessing

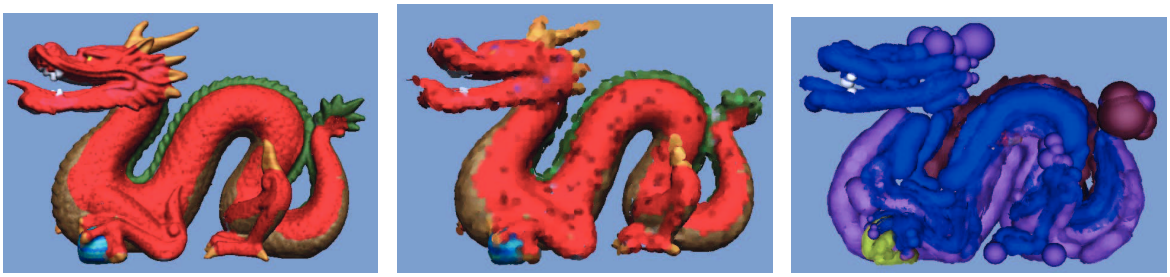
The preprocessing step is divided into two parts, the extraction of the surface points and the creation of the multiresolution representation. The time for the extraction of the surface points is strongly related to the type and complexity of the object's geometry. Setting up the multiresolution representation is fast, it only depends on the number of extracted surface points and is in $O(nlogn)$.

As our texturing technique can be applied to surfaces of objects from arbitrary type, the example objects in this paper have different underlying object representations. For the polygonal mesh of the Stanford Dragon, extracting the 437,645 surface points (Figure 6(b)) is done instantaneously like for the 38,619 surface points of the dinosaur statue (Figure 4) by directly using the vertices of the mesh. For the implicit surface of the 3D ant defined by an FRep model, it took 46 seconds to extract the 140,616 surface points (Figure 8(g)) due to the complexity of the implicit formulation of the object. Extracting the 78,499 surface points (Figure 7(a)) from the well known Siemens head sampled on a 150x200x192 voxel grid took 13 seconds.

Setting up the multiresolution representation took less than a second for every example shown in this paper.

4.3 Real-time process

There are two major bottlenecks which determine the interactivity of our texturing approach. First, the evaluation of the constructive texturing tree to determine the photometric attributes of the surface points when nodes to the attribute tree were added, and second, the rendering of the surface points. Thanks to the multiresolution approach we use, maintaining interactivity for both bottlenecks is achieved by choosing the right balance between the number of surface points which can be evaluated by traversing the constructive texturing tree and the number of surface points which can be rendered.



(a) high resolution (b) very low resolution (c) corresponding space partitions

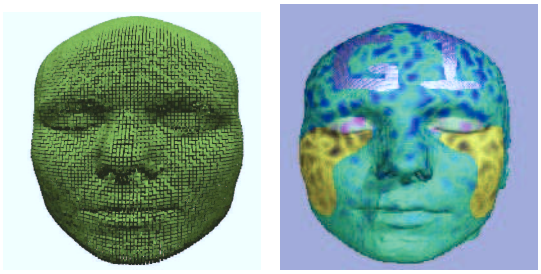
Figure 6: A more complex example using multiple space-partitions.

The time to determine the photometric attributes of the surface points heavily depends on the complexity of the solid texture described by the constructive texturing tree. The evaluation of the photometric attributes at the surface points becomes critical only when very complex primitives are used. In all the examples shown in this paper, the photometric attributes of all surface points as well as of the interior nodes of the hierarchy used for multiresolution could be evaluated in less than a second.

If the photometric attributes of a huge number of surface points are determined, simple hardware splatting of the surface points suffices [Rusin00], resulting in high-quality images when the projection of the bounding sphere associated to each surface point falls only on a few screen pixels. Using our rendering implementation, we can render up to 5M points per second using a Geforce 3 graphics board.

If only a small number of surface points could be evaluated, a high-quality software splatting technique is used for point-based rendering. The adapted level-of-detail of the multiresolution representation is chosen to insure interactive framerates. Using our rendering implementation of the output-sensitive surface splatting technique, we can render up to 300k surface splats per second in a 512x512 window. Note that the authors of the surface splatting technique claim to render up to 500k surface splats per second using a better optimised implementation on a similar hardware configuration [Zwick01]. However, we envisage to use the hardware-accelerated approach of [Ren02], which is less output-sensitive and where up to 3M surface points per second can be rendered. Moreover, we recently became aware of the hardware-accelerated point-based multiresolution rendering approach [Dachs03] which is perfectly suited for our approach. By shifting the computational cost to traverse the multiresolution hierarchy from the CPU to the graphics hardware, CPU load is liberated for evaluating the photometric attributes while rendering over 50M points per second.

The desired quality/speed trade-off can be chosen, Figure 8 shows different levels-of-detail of our 3D ant rendered using surface splats (Figures 8(a)-8(c)) and hardware splats (Figures 8(d)-8(f)), as well as the obtained framerates.



(a) 78,499 extracted discrete surface points (b) ray-traced image

Figure 7: Texturing a surface from the Siemens voxel array head sampled on a 150x200x192 grid.

4.4 User interaction

A more complex textured object, the Stanford Dragon described by 437,645 points, can be seen in Figure 6(a). We also show another resolution that was used during the texturing step (Figure 6(b)) as well as the corresponding space partitions that have been established (Figure 6(c)).

4.5 Postprocessing

In our environment, we used PovRay [PovRa] to obtain photorealistic rendering. Some of our results are shown in Figure 7(b) and Figure 8(h).

5 CONCLUSIONS

In this paper, we presented a new idea that links the texture and the geometry of an object by combining two approaches previously developed in computer graphics: *constructive texturing* and *point-based rendering*. This combination allowed us to develop a software environment where 3D objects of arbitrary nature (i.e. polygonal meshes, isosurfaces of voxel arrays, parametric or implicit surfaces, and others) can be textured by using an interactive and intuitive process. An interactive framerate is always guaranteed, whatever the complexity of the geometry and/or the texture, because the environment uses a multiresolution representation of discrete surface points extracted from the ob-

ject, that is tuned according to the performance of the graphics hardware and according to the texture complexity. This multiresolution representation offers also high-quality antialiased point-based rendering. One major advantage of our approach is that point-based rendering is only used during the interactive texturing step. We always keep a feedback to the initial geometric representation of the object (polygonal mesh, parametric or implicit surface, or whatever) which means that the final textured object can be easily exported to standard graphics software that cannot directly handle discrete surface points (e.g. CAD systems, photorealistic rendering engines).

Our implementation is still under development: currently four kinds of FRep primitives (i.e. spheres, blocks, convolution surfaces, and radial basis functions applied to a set of discrete surface points) can be used to create a partition of the constructive texturing tree. An immediate extension will be to widen the set of available FRep primitives. Two more straightforward extensions will be to integrate the hardware-accelerated high-quality splatting technique [Ren02] that renders more than 3M points per second on current graphics hardware, and the sequential point trees technique [Dachs03], that shifts the CPU load for the multiresolution rendering to the graphics hardware.

We are also investigating two promising directions. The first one is to allow the texturing of geometric attributes in addition to photometric ones, such as bump mapping or displacement mapping. Such a feature is offered in the Pointshop3D environment as it works easily exclusively on a cloud of points. In our case, as we want to keep a feedback to the original geometric representation, the process is not that obvious. One solution that we are currently implementing is to use the implicit deformation technique proposed by Cani [Cani93]. The second direction that we are investigating is to use this interactive kernel to manipulate something else than photometric or geometric attributes. By covering the object with a set of parameterised primitives, one can build a set of local parameterisations of the surface that can be smoothly blended together and used as a support for conventional hardware texture mapping.

ACKNOWLEDGEMENTS

We are grateful to Tamy Boubekeur, Clment Bichel, Blanca Borro Escribano, and Elisabeth Brunet for the design and implementation of the Pointshop3D plugin.

REFERENCES

[Adzhi99] Valery Adzhiev, Richard Cartwright, Eric Fausett, Anatoli Ossipov, Alexander Pasko, and Vladimir V. Savchenko. Hyperfun project: A framework for collaborative multidimensional F-rep modeling. In *Proceedings of the Implicit Surfaces 99*, pages 59–69, 1999.

[Agraw95] Maneesh Agrawala, Andrew C. Beers, and Marc Levoy. 3D painting on scanned surfaces. In *1995 Symposium on Interactive 3D Graphics*, pages 145–150, April 1995.

[Alexa01] Marc Alexa, Johannes Behr, Daniel Cohen-Or, David Levin, Shachar Fleishman, and Claudio T. Silva. Point set surfaces. In *IEEE Visualization 2001*, pages 21–28, October 2001.

[Benso02] David Benson and Joel Davis. Octree textures. *ACM Transactions on Graphics*, 21(3):785–790, July 2002.

[Cani93] Marie-Paule Cani. An implicit formulation for precise contact modeling between flexible solids. In *Proceedings of ACM SIGGRAPH 93*, pages 313–320, August 1993.

[Dachs03] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. *ACM Transactions on Graphics*, 22(3), 2003.

[DeBry02] David (grue) DeBry, Jonathan Gibbs, Devorah DeLeon Petty, and Nate Robins. Painting and rendering textures on unparameterized models. *ACM Transactions on Graphics*, 21(3):763–768, July 2002.

[Gross98] J. P. Grossman and William J. Dally. Point sample rendering. In *Rendering Techniques 98*, pages 181–192. Springer Verlag, 1998.

[Hanra90] Pat Hanrahan and Paul E. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, 24(4):215–223, August 1990.

[Levoy85] Marc Levoy and Turner Whitted. The use of points as display primitive. Technical Report TR 85–022, University of North Carolina at Chapel Hill, 1985.

[Lindh01] Erik Lindholm, Mark J. Kilgard, and Henry Moreton. A user-programmable vertex engine. In *Proceedings of ACM SIGGRAPH 2001*, pages 149–158. ACM Press / ACM SIGGRAPH, August 2001.

[Maya] Maya. *Alias-WaveFont*. www.aliaswavefront.com.

[Pasko95] Alexander Pasko, Valery Adzhiev, Alexei Sourin, and Vladimir V. Savchenko. Function representation in geometric modelling: concept, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995.

[Pauly03] Mark Pauly, Richard Keiser, Leif Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. *ACM Transactions on Graphics*, 22(3), 2003.

[Peder95] Hans Køhling Pedersen. Decorating implicit surfaces. In *Proceedings of ACM SIGGRAPH 95*, pages 291–300, August 1995.

[Pfist00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000*, pages 335–342, July 2000.

[PovRa] PovRay. *The Persistance of Vision*. www.povray.org.

[Praun00] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *Proceedings of ACM SIGGRAPH 2000*, pages 465–470, July 2000.

[Ren02] Liu Ren, Hanspeter Pfister, and Matthias Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum (Eurographics 2002)*, 21(3):461–470, 2002.

[Reute03] Patrick Reuter, Ireneusz Tobor, Christophe Schlick, and Sebastien Dedieu. Point-based modelling and rendering using radial basis functions. *Proceedings of ACM Graphite 2003*, February 2003.

[Rusin00] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, pages 343–352, July 2000.

[Savch95] Vladimir V. Savchenko, Alexander Pasko, Oleg G. Okunev, and Tosiyasu L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, October 1995.

(a) 140,616 high-quality surface splats rendered at > 1.5 fps (frames per second)

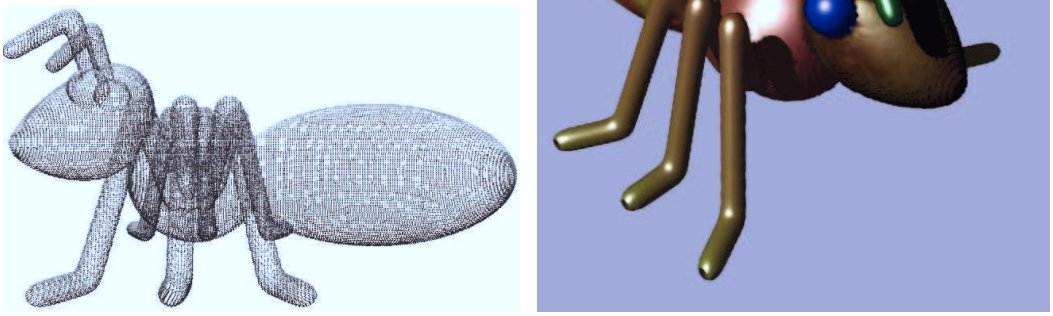
(b) 34,980 high-quality surface splats rendered at > 3 fps

(c) 11,043 high-quality surface splats rendered at > 5 fps

(d) 140,616 hardware splats rendered at $>$ 30 fps

(e) 34,980 hardware splats rendered at > 95 fps

(f) 11,043 hardware splats rendered at $>$ 200 fps



(g) 140,616 extracted discrete surface points from FRep model

(h) ray-traced image

Figure 8: Multiresolution rendering using hardware splats and high-quality surface splats and obtained framerates, starting from 140,616 discrete surface points extracted from an FRep model, and the final ray-traced image.

[Schmi01] Benjamin Schmitt, Alexander Pasko, Valery Adzhiev, and Christophe Schlick. Constructive texturing based on hypervolume modeling. *The Journal of Visualization and Computer Animation*, 12:297–310, 2001.

[Szeli92] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, 26(2):185–194, July 1992.

[Turk99] Greg Turk and James O'Brien. Shape transformation using variational implicit functions. In *Proceedings of ACM SIGGRAPH 99*, pages 335–342, August 1999.

[Turk02] Greg Turk and James F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics*, 21(4):855–873, 2002.

[Wei01] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of ACM SIGGRAPH 2001*, pages 355–360, August 2001.

[Witki94] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of ACM SIGGRAPH 94*, pages 269–278, July 1994.

[Zonen98] Ruben Zonenschein, Jonas Gomes, Luiz Velho, and Luiz Henrique de Figueiredo. Controlling texture mapping onto implicit surfaces with particle systems. In *Proceedings of Implicit Surfaces '98*, pages 131–138, June 1998.

[Zwick01] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *Proceedings of ACM SIGGRAPH 2001*, pages 371–378, August 2001.

[Zwick02] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3D: An interactive system for point-based surface editing. *ACM Transactions on Graphics*, 21(3):322–329, July 2002.