TOWARDS INTERACTIVITY ON TEXTURING IMPLICIT SURFACES: A DISTRIBUTED APPROACH

R. Zonenschein[1,2] **J. Gomes**[2] **L. Velho**[2] **N. Rodriguez**[1]

[1]Department of Computer Science, Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente 225, 22453-900 Rio de Janeiro, RJ, Brazil
ruben | noemi@inf.puc-rio.br

[2]IMPA – Instituto de Matemática Pura e Aplicada, Rio de Janeiro
Estrada Dona Castorina 110, 22460-320, Rio de aneiro, RJ, Brazil
ruben | jonas | lvelho@visgraf.impa.br

ABSTRACT

We describe a distributed system for texture mapping implicit surfaces. The method uses a particle system associated with the gradient vector field of the function that defines an implicit surface to acquire texture coordinates at a support surface. As each particle trajectory is independent of each other, this method has a special suitability to run in parallel. Our results show good improvement as more processors are added to the system, with speedups of up to 13 with 32 processors, performance such as required by an interactive system.

Keywords: implicit surfaces, texture mapping, parallelism

1 INTRODUCTION

Texture mapping is a successful technique in computer graphics. It maps a texture source onto an object increasing surface detail while maintaining the underlying geometry. While these are some properties greatly desired in computer graphics systems (fine details and simple geometry), in order to be successfully used, texture mapping demands interactivity, requiring a good degree of performance.

The texture source, known as a *texture map*, may be described by an image, a look-up table or a mathematical function. They may then be classified in two types, depending on their core nature, either 2D or 3D. This classification yields two variations of texture mapping techniques: 2D texture mapping [Barr83a], which gives the idea of sticking a label onto an object surface, or wrapping it with a 2D layer; and solid texturing [Peach85a, Perli85a, Wyvil87a], which gives the idea of carving an object out of a 3D structure such as wood or marble. In both cases, texture mapping is basically a matter of associating a texture space to the surface space, where the texture will be applied.

In the case of solid texturing, a 3D texture space is embedded in the object space and texture coordinates values are thus defined at all surface points. Although limited to textures that have a 3D structure, solid textures has the advantage that it can be applied with success to any model in 3D, independently of its description.

2D texture mapping is specially suited to parametrically described surfaces, since the mapping of two parametric spaces is usually straightforward. Implicit surfaces [Blinn82a], i.e, those defined by an iso-contour of a distance function, present an intrinsic difficulty for traditional 2D texture mapping: implicit surfaces do not have a natural coordinate system defined on them, therefore, association of texture coordinates to surface points cannot take place. 2D texture mapping of implicit surfaces has then relied on parameterization techniques, which is not always possible.

We have devised a global method [Zonen95a, Zonen97a] that allows one to apply 2D textures onto implicitly defined objects. Our method uses an intrinsic property of implicit models, the gradient vector field of the function that defines the implicit surface, to generate a force field. This is then used to simu-

late a particle system that maps the implicitly defined model to a support surface holding the texture.

Implicit surfaces and 2D textures have been a topic of intensive research. Pedersen[Peder95a] presented a method that performs local parametrizations creating patches over it. The patches can then be textured and interactively manipulated. Smets-Solanes[Smets96a] described a method that constraints a particle system to lie on the implicit surface. Assuming that a texture is already applied onto the surface, his method allows one to animate the texture while maintaining it on the surface. For a detailed comparison of their methods and the one presented in this paper, we refer the reader to our paper [Zonen98b].

The ultimate goal of a texture mapping system is to provide tools for the user to interactively map a texture onto a model and continuously refine it. Our method has proved to be able to provide such rich set of tools for texture mapping control [Zonen98a]. Moreover, the potential of this method also encouraged us to further generalize it to be used on composite deformable implicit objects [Zonen98b].

Although these improvements confirmed the usefulness of our method, they also highlighted the necessity of better iteration performance. Complex and deformable models involve large number of particles participating in our simulation, increasing the computational cost. Interactive control over this particle system requires faster computation methods.

We have been working on two distinct approaches to achieve the degree of interactivity our system should have: multiresolution and parallelism. In this paper we exploit an intrinsic suitability of this method to run in, and gain the most of, a distributed system. While most computer graphics applications use the parallelism paradigm for batch rendering, we use it to increase user interactivity. We haven't found publications on parallelism applied specifically to texture mapping.

2 PARTICLE TEXTURING OF IMPLICIT SURFACES

An implicit surface is defined as the set of points x in space that satisfy an equation $F(x) = c$, where $F: R^3 \rightarrow R$ and $c \in R$. Thus, F defines a continuous family of *level surfaces*, one for each *isovalue* c. The level surfaces of F follow the gradient vector field ∇F orthogonally, as can be seen in the Figure 1. This observation is the starting point for a method for sampling implicit surfaces [deFig96a] and for the texturing method described in this section.

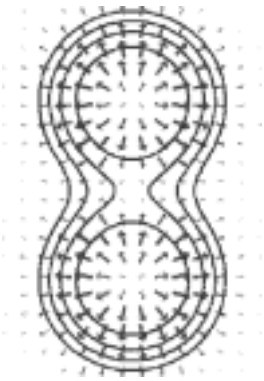

Figure 1: Gradient field follows level surfaces.

We interpret ∇F as a force field, which we use to drive a particle system. Let S be an implicit surface defined by a function F. We employ the gradient vector field ∇F to generate a force field defined in the ambient space, and we use it to guide particles that are initially at rest on S. The motion of a particle is governed by the differential equation

$$\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} + \nabla F = 0,$$

where x is the position of the particle, t is time and γ is a viscosity constant.

This establishes a correspondence between points on the implicit surface and points on a support surface T, where a 2D texture is defined beforehand: the texture attribute for each point on S is taken from the intersection of the corresponding particle trajectory with T. This projection mapping method can be classified as a two-step texture mapping technique[Bier86a]. While the projection step associates the impicit surface to the support surface via particle simulation, the mapping step associates the 2D texture to the support surface. Figure 2 shows the steps of this process and Figure 7 illustrates the results of the algorithm in 3D.

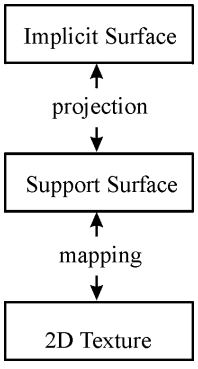

Figure 2: The method and its steps

The pseudo-code in Algorithm 1 shows the inner loop of the particle system simulation. Although better performance can be achieved using an adaptive Range-Kutta integration, for the sake of simplicity we present the most simple computation of a particle trajectory along the gradient vector field using a direct Euler method.

Algorithm 1 Particles simulation inner loop

```
for each particle p at S
    while p does not intersect T
        p = p + dt∇F(p)
```

Although computationally simple, the amount of time needed to run the particle system may vary depending on various factors, such as:

- *Spatial relationship*: the proximity between S and T.
- *Time step*: time step (dt) chosen for the simulation.
- *Model complexity*: number of particles participating in the simulation.

Moreover, fast computation is extremely desired when interactivity is a goal. We have designed an interactive control system that takes advantage of the particle texturing method providing tools for the user to manipulate the texture placement in various ways. This system, described in depth in [Zonen98a], introduces global and local control tools, using the geometric, parametric and temporal characteristics of the method. Interacting with this system, a user can manipulate different parameters and introduce other forces in the medium while globally visualizing the results. Some of the control parameters are:

- Time step (dt).
- The weight of contribution to the field between ∇F and ∇T (∇T contributes to the field attracting the particles to T).
- The function parameters of external forces, used mostly for local changes.
- The position of the basic elements S, T, and external forces. Any slight change on their positions affects the texture placement.

Even though we rely on hardware with texture-dedicated engines, real time interactivity can only be achieved with the help of other computation schemes. In the following section we describe our approach for applying the particle system method in a distributed system.

3 DISTRIBUTED SYSTEM FOR PARTICLE TEXTURING

The particle system described in the previous section has a special suitability to run on distributed processors. Each particle has its trajectory depending only on the forces applied to it, without any interference with other particles. This is thus an *embarrassingly parallel problem* [Foste95a], where computation consists of a number of tasks that can execute independently, without communication.

We have used the LAM implementation of the MPI communication standard as a programming environment to develop a client-server version of our particle system method. We have taken the independency property of particles to assign the simulation in chunks to different processors. The following three pseudocodes resume this approach.

Algorithm 2 first initializes the MPI environment. This amounts to the basic MPI routines of taking the number of processors participating in the distributed system and the processor *id*. It then initializes the particle system, reading files and setting parameters for the simulation. Depending on the MPI *id* assignment, the processor is switched to the corresponding code, behaving as a master or a slave. When the master and slaves work are done, the textured model can then be visualized.

Algorithm 2 Main

```
Initialize MPI
Initialize particle system
if (id == 0)
    Run master
else
    Run slave
Visualize
```

The main duty of master algorithm 3 is particle assignment. It divides the total number of particles in equal parts, and distributes the resulting sets among the processors, assigning the remainder of the division to the last processor. For each slave, it then sends an MPI message with their first and last particles assignment. As we will describe in section 5, different strategies for particle assignment may be used when working with multiresolution models. The algorithm finalizes its work waiting for a message of completion from each slave.

Algorithm 3 Master

```
Calculate number of particles per slave
For each slave
    Send message with First and Last particles
Wait for completion
Exit slaves
```

Slave algorithm 4 waits to receive a message with the first and last particles representing its scope of action in the assignment. After processing the particles simulation, as in algorithm 1, it then sends a message of completion to the master. The confirmation is received via a *die tag* message, when the processor frees itself.

Algorithm 4 Slave

```
Receive message from master
If message contains First and Last particles
    Simulate particles from First to Last
    Send message DONE to master
If message is DIETAG
    Exit
```

It should be noted that the code presented above has a cost of initializing the particle system at each processor. This can be highly time consuming as the geometry data of a 3D model may be too large. This apparent pitfall is overcomed when applying the algorithm to behave in an interactive way, as desired by a controllable texture mapping system. Moreover, there are very few data being passed between processors. Particle assignments from master to slaves are done only once per slave, using two integers, one for the first and other for the last particles. The parameters that the user may vary do not impose much communication overhead either, as the distance function F used for the simulation is very compact. In this interactive system, each particle's (u, v) texture coordinates can be passed straight to the visualization module. Algorithm 5 describes the interactive steps of the slaves processors.

Algorithm 5 Interactive slave

```
Receives message with First and Last particles
While message received is not DIETAG
    Get system parameters
    Simulate particles from First to Last
    Sends data to Visualization module
```

4 RESULTS

Our tests were run in a dedicated Linux cluster with 32 machines in a 10 Mbits/sec network. Table 1 shows the results we obtained using three different models, with 262, 530 and 2558 particles with timestep of 0.0078. Taking the runtime on one machine as a parameter, we calculate both speedup (S), the factor by which execution time is reduced on p processors, and efficiency (E), a percentage of the desired "one processor" runtime when running in p processors. This amounts to the following formulae[Kumar94a]:

$$E = \frac{T1}{p * Tp},$$

where Tp is the runtime obtained with p processors, and

$$S = \frac{T1}{Tp} = E * p.$$

Particles	p	Time(s)	Speedup	Efficiency
262	2	0.214	1.59	0.79
	4	0.134	2.53	0.63
	8	0.080	4.23	0.53
	16	0.048	6.99	0.43
	24	0.031	10.80	0.45
	32	0.025	13.65	0.44
530	2	0.477	1.50	0.75
	4	0.272	2.63	0.66
	8	0.203	3.53	0.44
	16	0.107	6.70	0.41
	24	0.073	9.77	0.40
	32	0.056	12.59	0.40
2558	2	2.315	1.49	0.75
	4	1.223	2.82	0.70
	8	0.918	3.76	0.47
	16	0.562	6.14	0.38
	24	0.377	9.14	0.38
	32	0.287	12.00	0.38

Table 1: Performance, speedup and efficiency related to sequential runtime: 0.341, 0.718 and 3.454 (s) respectively to 262, 530 and 2558 particles

The system performance is better viewed in the graphs in Figures 3 and 4. As more processors are added to the system, higher speed is obtained. The speedups are plotted in Figure 5, and the system efficiency is plotted in the graph in Figure 6.

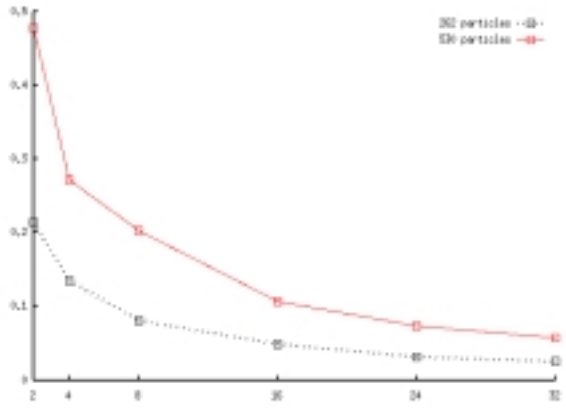

Figure 3: Performances with 262 and 530 particles

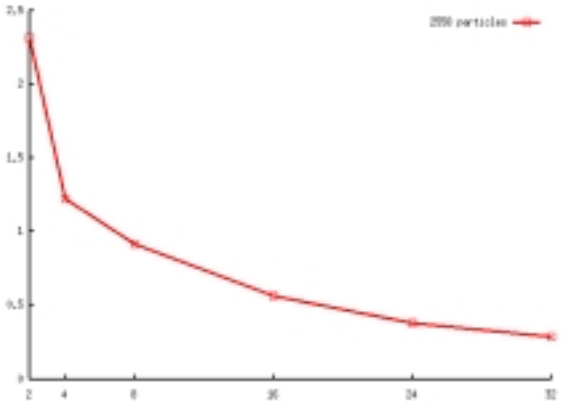

Figure 4: Performance with 2558 particles

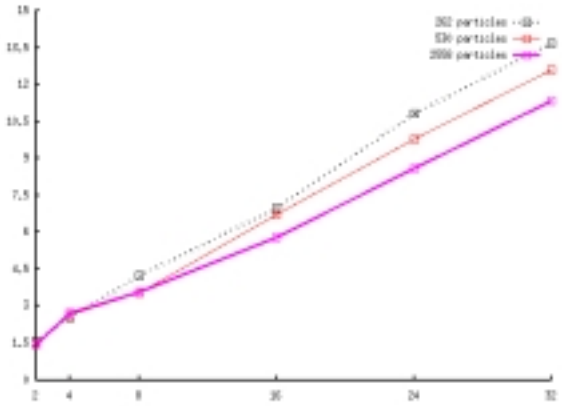

Figure 5: Speedups

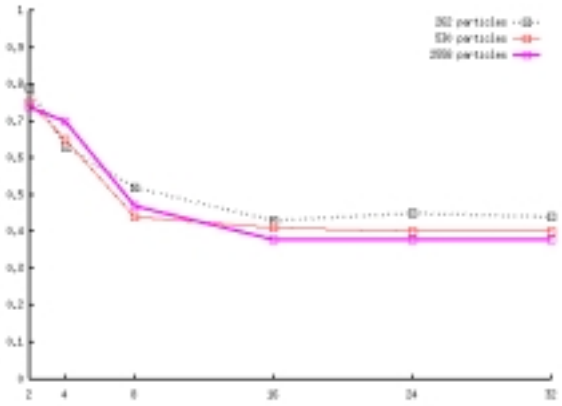

Figure 6: Efficiency

As the visualization module does not present a bottleneck in our system architecture ((u, v) texture coordinates of each particle do not impose communication overhead) we can gain the most of the speedups obtained. Although the efficiency was expected to decrease as more processors participate in the simulation, their values prove the effectiveness of such system.

Load balance is obtained taking advantage of the geometry proximity of particles at their starting points on the model. As we use a polygonized implicit model, we can be sure that all particles on a polygon are close to each other, and so are each consecutive polygon. Because closer particles commonly have similar trajectories along the simulation, load balance is guaranteed assigning them to different processors.

5 CURRENT WORK

We are currently working on a system that combines multiresolution models and the distributed system presented. A multiresolution model is basically a 3D object described in different levels, from the most coarse to a very refined one. In our texturing system, this implies varying the number of particles participating in the simulation, with texture placement quality accompanying the geometry level. We believe that our interactive system can greatly improve in speed and functionality with this combination.

We have set our particle system to run in a continuous fashion. Basically, when a particle reaches the support surface T, (u, v) texture coordinates can be passed to the display module, and the particle is free to restart its simulation at S. We interpret this as a starting point for a new resolution level of particles to be simulated. While no user interaction takes place, finer resolution levels are continuously simulated and their results displayed over the prior level. When the user interacts with the system, the most coarse level of resolution is computed again, restarting the consecutive levels' simulations. In fact, depending on the scope of the control tool used by the user, only parts of the model should be driven to the coarse level. This is particularly relevant when a local control tool is used. In this case, we compute the range of action of this tool and apply the coarse level simulation only on this range. The remaining parts of the model maintain their resolution levels path.

We can take more advantage of the multiresolution approach implementing it in a distributed system. Particle assignment among processors are precalculated involving all resolution levels. As each processor knows which particles it should compute at each level, system parameters settings define what each processor should do at a given time. Slave algorithm 6 can thus be rewritten to accommodate multiresolution processing.

Algorithm 6 Interactive slave for multiresolution

```
Receive message with First and Last particles
While message received is not DIETAG
    For each resolution level
        Get system parameters
        Simulate particles for this level, if applicable
        Send data to Visualization module
```

From the user's point of view, two desired characteristics of an interactive texture mapping system are then achieved. One is the basic motivation of this paper: faster computation. The other one is a very good feedback degree. At any moment, the user is prompted with a result of its texture placement. Even though coarse in the begining, the user may then decide if an interaction is needed, using either global or local control tools. At all times, the system is continuously being updated with better resolution levels.

6 ACKNOWLEDGEMENTS

This research has been developed in the VISGRAF laboratory at IMPA, as part of the PhD program of the first author at the Computer Science Department of PUC-Rio. VISGRAF laboratory is sponsored by CNPq, FAPERJ, FINEP and IBM Brasil. We would like to thank Roberto de Beauclair Seixas, from IMPA, for his advice on analyzing the results.

REFERENCES

[Zonen95a] Zonenschein, R., Gomes, J., Velho, L., de Figueiredo, L. H.: Textura de superfícies implícitas com sistemas de partículas", *Proceedings of SIBGRAPI'95 (Brazilian Symposium on Computer Graphics and Image Processing), in Portuguese*, pp. 305–306, 1995.

[Zonen97a] Zonenschein, R., Gomes, J., Velho, L., de Figueiredo, L. H.: Texturing implicit surfaces with particle systems, *SIGGRAPH'97 Visual Proceedings*, p. 172, 1997.

[Zonen98a] Zonenschein, R., Gomes, J., Velho, L., de Figueiredo, L. H.: Controlling texture mapping onto implicit surfaces with particle systems, *Workshop on Implicit Surfaces '98*, pp. 131–138, 1998.

[Zonen98b] Zonenschein, R., Gomes, J., Velho, L., de Figueiredo, L. H., Tigges, M., Wyvill, B.: Texturing composite deformable implicit objects, *Proceedings of SIBGRAPI '98 (Brazilian Symposium on Computer Graphics and Image Processing)*, pp. 346–353, 1998.

[Barr83a] Barr, A.: Decals, *State-of-the-Art of Image Synthesis*, SIGGRAPH'83 Course Notes, 1983.

[Bier86a] Bier,E. A. and Sloan Jr, K. R: Two Part Texture Mappings, *IEEE Computer Graphics and Applications*, Volume 6, Number 9, 1986, pp. 40–53, 1986.

[Blinn82a] Blinn, J. F., A generalization of algebraic surface drawing, *ACM Transactions on Graphics*, 1(3), pp.235–256, 1982.

[deFig96a] de Figueiredo, L. H., Gomes, J.: Sampling implicit surfaces with physically-based particle systems, *Computer & Graphics*, 20(3), pp. 365–375, 1996.

[Foste95a] Foster, I.: Designing and Building Parallel Programs, *Addison Wesley*, 1995.

[Kumar94a] Kumar, V., and others: Introduction to Parallel Computing, *Benjamin/Cummings*, 1994.

[Peach85a] Peachey, D.: Solid texturing of complex surfaces, *Proceedings of SIGGRAPH'85*, pp. 279–286, 1985.

[Peder95a] Pedersen,H.K.: Decorating implicit surfaces, *Proceedings of the SIGGRAPH '95*, pp. 291–300.

[Perli85a] Perlin, K.: An image synthesizer, *Proceedings of SIGGRAPH'85*, pp. 287–294, 1985.

[Smets96a] Smets-Solanes,J.P.: Vector field based texture mapping of animated implicit objects, *Eurographics '96 Conference Proceedings*, Poitier, France, 1996.

[Wyvil87a] Wyvill. G., Wyvill, B., McPheeters, C.: Solid texturing of soft objects. *IEEE Computer Graphics & Applications*, 7(12), pp. 20–26, 1987.

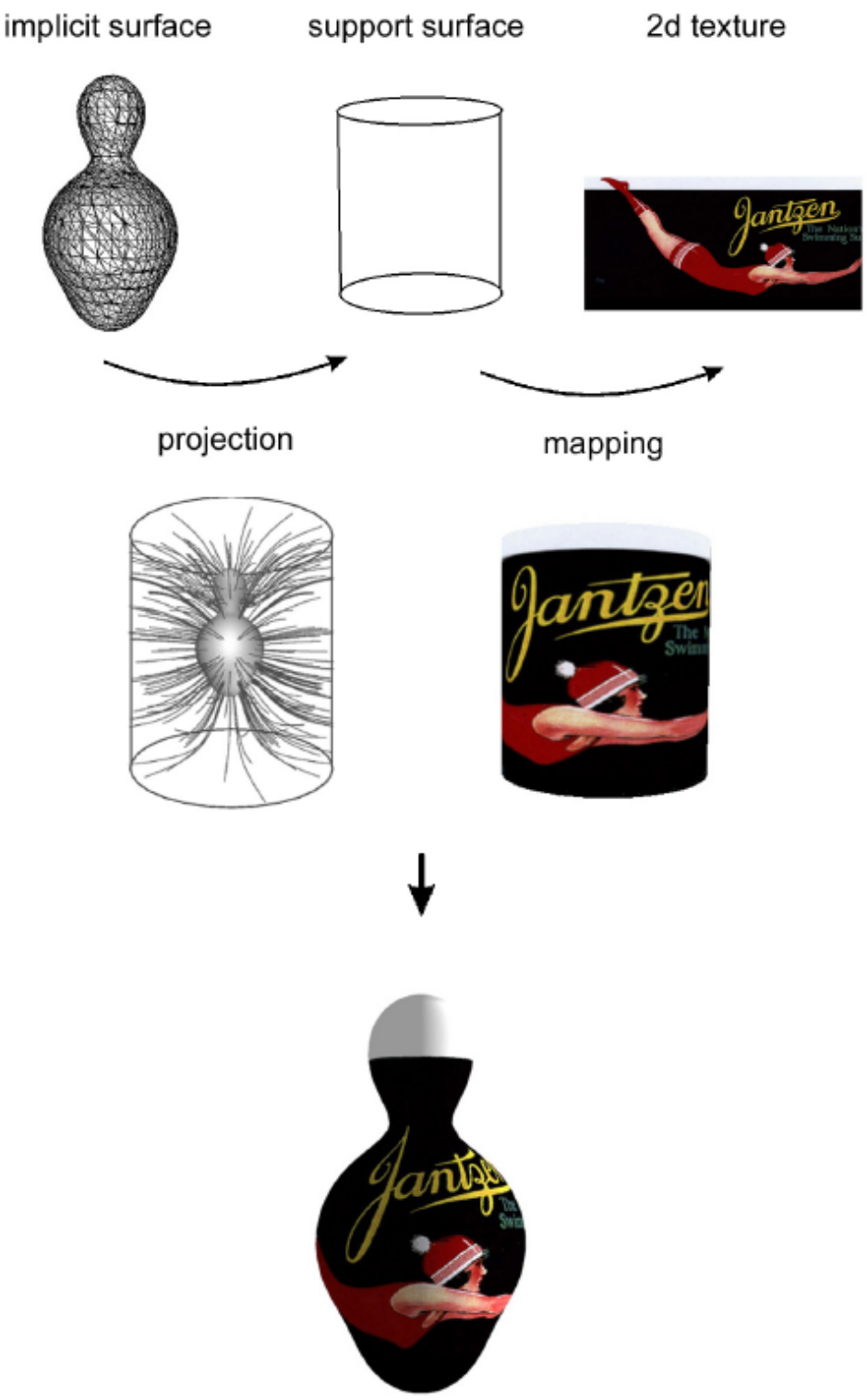

Figure 7: An overview of the texture mapping method