# Realistic Rendering of an Organ Surface in Real-Time for Laparoscopic Surgery Simulation


Raphaël Heiss, Fabrice Neyret and Franck Sénégas
e-mail: Fabrice.Neyret@imag.fr

iMAGIS[†] / GRAVIR-IMAG

contact address : Fabrice Neyret
iMAGIS - GRAVIR / IMAG
INRIA Rhône-Alpes ZIRST
655 avenue de l'Europe
38330 Montbonnot Saint Martin
FRANCE


[†]iMAGIS is a joint research project of CNRS / INRIA / UJF / INPG.

## Abstract

This paper deals with the rendering issues of the problem of producing a real-time convincing surgery simulation. The main scope of our project is to simulate laparoscopic liver surgery. Nevertheless large parts of the technique apply to other organs. We address three aspects of the appearance of the organ surface: the organ skin texture, the specular highlights, and the reactions of the organ to the instruments.

For this last aspect we address three effects: blood drops rolling on the surface, clear or deep cauterization, and whitening of the surface under local pressure.
To meet the real-time constraint we use advanced graphics features such as multi-pass rendering, OpenGL texture extensions and lookup-tables. Our target hardware are high-end graphics accelerators such as the SGI Infinite Reality. Nevertheless most of the technique apply to low-end graphics accelerators.

**Key words :** Real-time rendering, multipass, simulator, medical applications, textures.

## 1 Introduction

**Why simulating ?** Laparoscopic surgery is a non-invasive technique that consists of introducing through small holes in the patient's abdomen several micro-instruments and an optic fiber connected to a camera and to a light source (see Fig. 1). This new technique is expanding, but it requires a lot of training for a physician to be proficient. Classical training sessions using cadavers or animals are costly, yield ethic problems, and cannot easily train for particular pathologies. The fact that laparoscopy surgery consists of looking at a monitor while manipulating the scissor-like end of instruments outside the patient should make it easier to replace the patient by a computer with force-feedback devices. There is thus a demand for laparoscopic simulators.

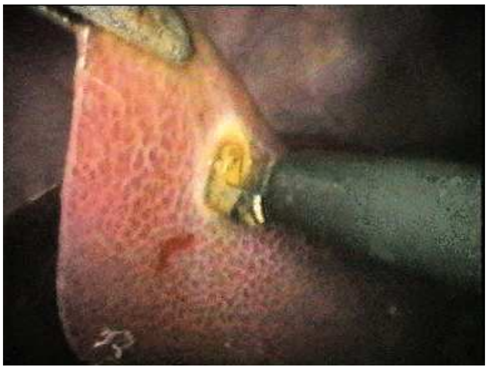

Fig. 1: A real liver laparoscopic image.

**Simulating what ?** Solving the problem of producing a real-time convincing simulation of the surgery of an organ requires that we address several issues:

- rendering in real-time the look of the organ surface, including the effects of the instruments and the reactions of the living organ;
- managing in real-time the collisions of the instruments with the organ;
- simulating the organ deformation in real-time.

This research is part of a larger project named AISIM (AISIM project ). The last two issues are being addressed by other groups within the project. See for instance (Debunne et al. 1999; Debunne et al. 2000; Cotin et al. 2000) for the simulation of deformations and (Lombardo et al. 1999; Picinbono et al. 2000) for the collisions management. In this paper, we only address the first issue.

**Why realistic rendering ?** In the scope of simulation for training professionals, the realism of the rendering is not a purpose in and of. It can even be dangerous if it provides non-pertinent informations to the trainee that he might use later in real situations. For instance, vessels may be used consciously or not to locate a site, while the vessel locations vary from one patient to the another. The presence of realism addresses three purposes: quality of immersion (it is important that the trainee reasonably trusts the simulation), carrying pertinent information, and providing 3D information that is present in the real situation (no more, no less, same modality).

To find the origin of the 3D sense mentioned above is not trivial, even during actual surgery: the monitor does not carry stereoscopic information, the depth of field of the camera is large, there is almost no shadow because the light source is mounted on the camera, and the shading variations are weak because of high inter-reflections (i.e. the ambient component of the illumination is high).

However, physicians need this 3D information to operate! The fact is that early trainees are depth-blind, and progressively acquire this perception, often without understanding where it comes from. Thus, several secondary clues provide pieces of 3D information. Some deal with the image, some not (e.g. contact feeling, a-priori knowledge of shapes). The firsts include the depth cue given by perspective shrinking of the textured surfaces, the highlights, that provide curvature information, which are also very important for the feeling of contact with the surface, and very thin shadows between distant and close objects.

Because most of the operation focuses on the main organ, we only deal with the first and the second issues.

**Rendering what ?** For providing 3D clues, we will map undistorted textures on the surface, and simulate the highlights due to the light source, which is a thin ring around the optic fiber.

The pertinent information to carry is the effects of the instruments on the surface and the live organ reactions, which include blood drops rolling on the surface, clear or deep cauterization, and whitening of the surface under local pressure.

To achieve high immersion quality, we will have to take care of realism when rendering these features. As the purpose is to obtain a simulator, all these have to be done in real-time. This suggests that we avoid increasing the geometry (e.g. for the blood drops), and use the advanced graphics features available on the graphics accelerator.

# 2 Previous Work

## 2.1 Texturing

Some surgery simulators show organs with constant color, or define colors only at the mesh vertices. The best aspect of organs is obtained by 'dressing' the mesh with textures.

Textures have been around for a long time in Computer Graphics (Catmull 1974). However, mapped textures have suffered from mapping distortion for a long time, despite various improvements. This constrains the designers to pre-distort the texture drawing in order to compensate the distortion during mapping. Several solutions to this problem have been proposed:

Direct painting (Hanrahan and Haeberli 1990) consists of painting the texture directly on the final surface and storing the color values in texture space. As the mapping distortion during the storage is the reverse of the one at rendering time, no distortion results (provided the mapping is bijective).

Undistorted pattern mapping using triangular tiles (Neyret and Cani 1999) solves the mapping problem for classes of textures that meet our requirements (isotropic and homogeneous). It consists of defining a small set (four) of compatible equilateral triangular patterns, to be associated to triangular areas defined by a regular triangulation of the surface. Mapping with tiles is interesting because it allows high image resolution using little texture memory, and it relies on samples of materials that are independent from objects shape.

Procedural solid textures, such as the ones introduced by Perlin (Perlin 1985) and Worley (Worley 1996), require no mapping. The second one produces textures made of cells, being based on the 3D Voronoï diagram of a Poisson distribution of points. This matches well the properties of biological tissues.

We combine somehow these three solutions for the various effects we want to obtain. We extend the first one to simulate the evolving effects. We rely on Worley

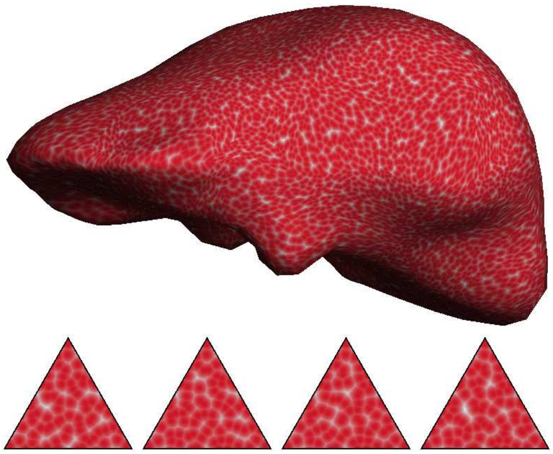

Fig. 2: *Top*: the liver texturing with minimal distortion. *Bottom:* the four triangular tiles used.

textures for the skin aspect. As we cannot afford a per-pixel procedural evaluation in real-time, we will precompute the texture. Fortunately, this has already been addressed for triangular tiles in (Neyret and Cani 1999) (see Figs. 2), thus we will base the skin rendering on this method.

Note that some work has been done on automatic texture acquisition. This may be used to obtain the texture of an organ from a set of photographs. However this reverse problem is especially hard in our case, as both the texture, the exact organ shape, the illumination conditions and the camera location relatively to the organ frame are unknown at the same time. Since satisfactory procedural textures can be produced as well, with the ability of getting compatible small patterns, we tend to prefer the easy solution.

## 2.2 Highlights

Reproducing the reflects of the light source is important in surgery simulation because they are a clue for depth, orientation and deformations. Organs are wet, and generally covered with a transparent tissue, thus being highly specular.

Specular highlights are generally smaller than the faces of the geometric mesh. Solutions based on Phong evaluation at the vertices thus yield a lot of specular aliasing (i.e. hexagonal spots, flickering shading). Moreover, Phong evaluation only considers a point light source. A convenient solution to these problems is to use reflection mapping (Miller and Hoffman 1984), to allow for reflections of the environment on the object. This fix the highlight resolution problem, and this allows for complex light source. Environment textures are available in hardware using regular textures. It consists of using for $(u, v)$ texture coordinates at each vertex the parametric location on the environment sphere that reflects at this vertex. Various improvements have been introduced to release the limitations of the model (i.e. view dependency, distant environment, motionless environment). For our application the simple solution is almost sufficient, as the environment texture simply represents the aspect of the specular spot, whose location is fixed and centered in the texture since the light and the view-point are merged. However, during surgery the light is not at infinity and its distance change, thus the spot aspect will have to evolve with the distance of the light. Alas, transferring a texture to a graphics board is slow, and doing it at each time step kills performances. We introduce a variant of this technique in order to process distance changes efficiently, and to take the surface roughness into account.

## 2.3 Advanced rendering features

Graphics libraries such as OpenGL (Neider et al. 1993) provide interesting features (hardware-accelerated on high-end graphics boards) allowing more and quicker effects. For instance, at that time we do not simulate other rendering aspects such as shadows or rough surface. If necessary, solutions could be found using shadow-maps and bump-mapping that can be obtained using advanced rendering features (Siggraph Course Notes 1998; Silicon Graphics b; Silicon Graphics a; Nvidia ).

The features we use to get our results and to match the real-time constraints are:

- multipass rendering, that allows us to add several layers of appearance on a surface;
- feedback rendering, which is the classical way to know what location of the geom-

etry is below the pointer;

- texture updating, that allows us to transfer only the part of the texture that has changed from one frame to the other;
- reflection mapping, as stated above;
- color look-up tables, allowing variations of the texture appearance without having to transfer the texture more than once.

### 2.4 Drops, burnt spots and whitened spots

Little work has been done on drops. Drops used in CG applications are often based on particle systems (Reeves 1983; Reeves and Blau 1985), associating a fake visible object (a disk, a 2D sprite, a line segment) with a physically animated point object. Tears have been simulated in (Fournier et al. 1998). We do not need so much realism in the visual aspect, and we don't want to add hundreds of polygons in our scene, but we need the same kind of animation depending on gravity, shape curvature and friction. For the other reactions of the organ, i.e. burning under cauterization and whitening under local pressure, no real motion occurs despite the evolving appearance, so no physical particle animation is needed. We will achieve the 'fake rendering' of all these objects by directly painting them in the texture at their simulated location, in a way analogous to (Hanrahan and Haeberli 1990), excepted that the painting is temporary.

## 3 Contributions

### 3.1 Our model

Our organ surface model consists of three layers to handle the three components of appearance, corresponding to three textures and to three rendering passes (see Fig. 3):

- the first one accounts for the organ skin appearance, including skin color, Lambert shading and skin texture. We rely on Worley patterns and the regular triangular tiles

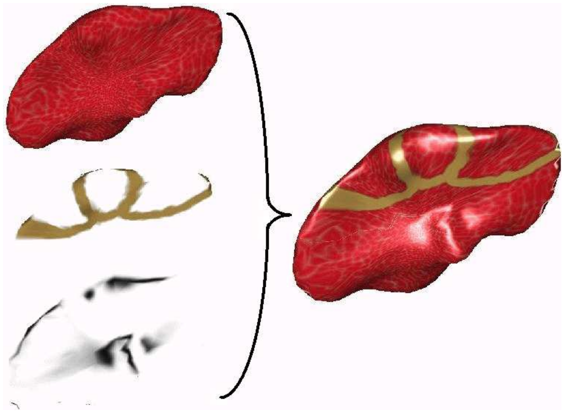

Fig. 3: The three texture layers: skin, reactions and reflects.

of (Neyret and Cani 1999), as illustrated on Fig. 2. We describe this in section 4.1.

- The second one deals with evolving reactions of the organ skin. These reactions are drawn dynamically in a transparent texture. We do this by extending the direct painting method (Hanrahan and Haeberli 1990), replacing the artist's brush by the simulation of the effects (blood drops, burning, whitening), getting the aspect shown on Fig. 4. The main issues to be solved are listed in sections 3.2 and 3.3; we detail the rendering in section 4.2 and the animation in section 5.
- The third one manages the highlights, using reflection mapping (Miller and Hoffman 1984), that we extended to take into account the variations of the light distance (see Figs. 6). We introduce the issues in section 3.4 and we detail our technique in section 4.3.

We give a pseudo-algorithm of the whole process in Appendix A.

## 3.2 Suppressing distortions

Distortions are avoided for the skin layer using the triangular tiles method. The reflects layer suffers no distortion, as the parametric space is based on polar coordinates, like the light source shape is. Thus we only have to deal with distortions of the reactions layer, which consists of a regular image texture.

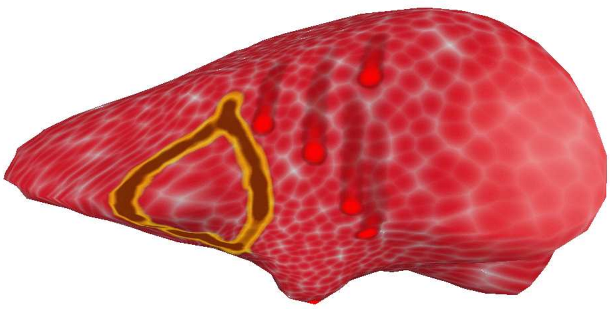

Fig. 4: The three kinds of dynamics effects: cauterization (left), blood drops (middle) and whitening (right).

We build dynamically this image on a way inspired by the direct painting method. But differently to the original method (Hanrahan and Haeberli 1990), we are drawing 'brushes' (i.e. 2D sprites) in the texture, and not simply point painting. Thus we have to take into account the mapping distortion in the neighborhood of the drawing location. This is achieved by considering the local Jacobian of the mapping: for a disk spot to appear on a location of the surface, an ellipse has to be drawn at the corresponding texture location. Details are given in section 4.2.3.

### 3.3 Evolving effects

The evolving effects are local (drops, spots, marks), and can be though as a kernel around an active spot (drop center, current pressure or heat site) followed by the fading 'memory' of the effect along the previous locations of the active spot. The kernel aspect, the fading nature and speed define the appearance of the effect, while the animation of the active spot defines its change with time.

We model these effects using snakes of sprites, much like the early computer game *Worms* in the eighties, except that our sprites are living in the texture space instead of the screen space (see Fig. 5). We developed an update strategy to minimize the texture transfer to the graphics board, knowing that only the sprite areas change from one frame to the other, and that the sprites overlap. The animation of these snakes is managed by

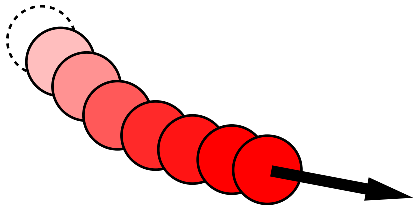

Fig. 5: An effect is represented by a snake of sprites in the texture space.

an automaton which handles the creation of new active spots, the fading of the old ones, the destruction of inactive spots, and collisions rules. Collisions (for drops) are handled using a list for each face that keeps track of the sprites present on the face (storing a sprite list per face is not memory consuming as there are both few faces and few sprites in the simulation). This mechanism also allows us to estimate the decrease of friction for a drop following another one (which will thus finally merge with it). The overlapping (i.e. collisions) between burnt paths should result in a strengthening of the burning. This is achieved by introducing an indirection: we keep in main memory a map of burning intensity, which is transformed into color using a lookup table.

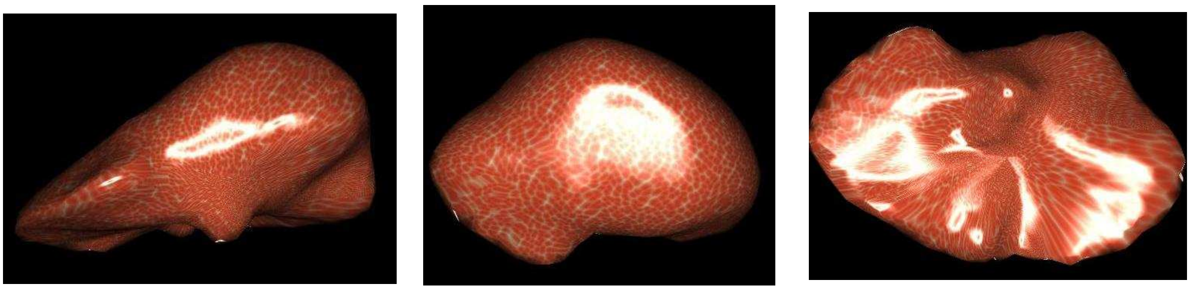

Fig. 6: Ring light source glossy reflection.

### 3.4 Specular effects

Specular spots on surfaces result from the combined effects of the Snell-Descartes perfect mirror reflection and the distribution of orientations that a rough surface has. We handle these two aspects separately in order to allow the tuning of light distance and surface roughness. We rely on environment textures, than can be hardware accelerated on any graphics board handling textures. As the highlight spot is axi-symmetric, we represent it with the composition of a constant radial ramp by the 1D profile of the reflect spot (see Figs. 14 and 15). This indirection allows us to only update the 1D table when the reflect aspect should change, instead of recomputing and transferring the whole 2D map. The 1D indirection table is implemented using hardware look-up tables. On hardware featuring this facility (such as the SGI Infinite Reality, or PC boards handling palette textures), this allows the real-time update of the highlight appearance.

# 4 The Three Texture Layers Liver Model

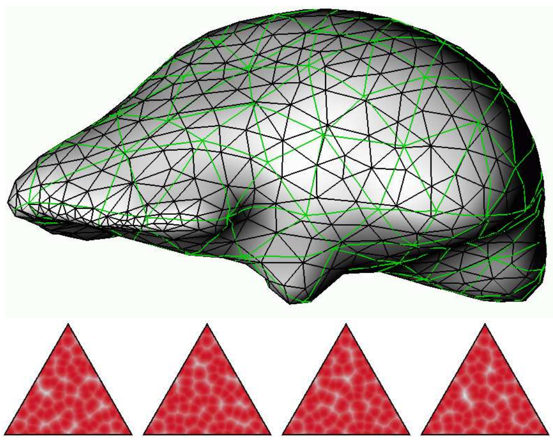

Fig. 7: *Top*: the texture mesh (in green) added to the geometric mesh. The extra vertices are handled as barycentric combination of the main vertices. *Bottom*: the triangular tiles to be mapped on the green mesh.

## 4.1 Skin Map

For this first layer, we apply the triangular patterns mapping method (Neyret and Cani 1999). The main difficulty lies in the fact that extra vertices are introduced in the geometry, that are the vertices of the texture mesh (figured in green on Fig. 7). This can yield a problem with the simulation of the organ deformation, as these extra vertices are not known by the deformation model and don't correspond to 3D elements inside the organ volume. Propagating the creation of surface vertices into the volumetric deformation model would increase the model complexity and the resolution time, for a reason unconnected to the simulation quality criteria, which is unacceptable. Instead, we consider these extra vertices as secondary vertices, defined at any time step as the barycentric combination of two or three primary vertices.

For the Worley (triangular) texture, we use only the closest neighbors (i.e. the regular Voronoï diagram of a Poisson distribution of points), and a simple color map to turn the distance into colors (see Fig. 7, Bottom).

## 4.2 Effects Map

The second layer is a transparent texture, where both the transitory and permanent reactions of the organ skin to the instruments actions are stored. This texture will permanently evolve. The problem is that transferring a texture to the graphics board is slow, so doing it at each time step can spoil the real time simulation. So we have to minimize the amount of data sent to the graphics board, by taking advantage of the locality of changes in the texture. Another issue is to draw the effects in such a way they do not appear distorted despite the mapping is.

### 4.2.1 Editing of the map

In main memory we maintain separately a transparent texture containing only the permanent effects: at each time step the effect texture in main memory is restored using this one, then all the active sprites are drawn by software in the effect texture. Then the effect texture on the graphics board is updated, by transferring only the parts that have changed as we explain below. If a sprite is not transparent when becoming inactive (essentially for the cauterization effects), it is drawn in the permanent effects texture.

### 4.2.2 Parameterization

Drawing dynamically the sprites in the texture is very similar to the direct painting methods, for which a designer paints the texture content directly at the surface of the object. In order to apply this method, we need to define a bijective mapping between the surface and the texture space. Several methods exist to deal with the general case. The organ we are working on (i.e. the liver) is close enough to a sphere so that the central projection of a sphere parameterization on the organ surface is bijective, if the sphere center is correctly located (so that the surface is star-shaped relative to this point). This parameterization creates locations with singularities needing special treatment: the poles and the 'date change line' joining the two poles. The 'date change line' treatment is similar to what has to be done in the classical case of torus topology: the part of the drawing that is outside one side has to be drawn on the other side (see Fig.10). The poles are more of an issue. We chose the parameterization in such a way that the poles come to two locations that are far from the potential operation areas (the vessels entry at the bottom of the liver, and the center of the surface on top).

### 4.2.3 Suppressing the distortions

The basic direct painting method (Hanrahan and Haeberli 1990) automatically cancels the distortions of the map assuming the texture pixels are drawn one by one, which is not our case (moreover, in the original method there are as many vertices as there are texture pixels). We have to pre-distort a sprite that is drawn in the texture in such a way that it will appear undistorted once mapped: a disk on the surface should be drawn in the texture as an ellipse, whose axis and radius corresponds to the eigenvectors and the eigenvalues of the Jacobian of the mapping at this location (see Fig. 8). In fact we don't need to explicitly construct this ellipse: a location $U$ in the neighborhood of the ellipse center $Uc$ in the texture space can be quickly converted into a location $P$ in the neighborhood of the disk center $Pc$ in the surface space using the Jacobian:

$(P - Pc) = J(U - Uc)$. The square distance from $P$ to $Pc$ indicates if the location is inside or outside the disk, and this distance can be used as a parameter of a function controlling the profile of color and opacity in this disk. The Jacobian also provides a bounding box of the ellipse (the two sizes are given by the norm of the two columns times the sprite diameter). So, to draw a sprite in the texture, we scan the bounding box area and proceed for each texture pixel as stated above (the factors in the transform can be computed incrementally).

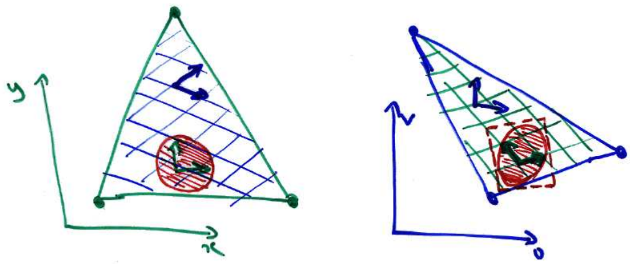

Fig. 8: Due to the mapping distortion, a disk on the surface appears as an ellipse in texture space.

As the mapping is linear within each face, the Jacobian is constant within each face (a part of the disk may exit the face, but we assume that the mapping distortion is not too different for this part of the disk). We could precompute and store the Jacobian for each face (this is not memory consuming as meshes used in surgery simulations are usually simple, due to the cost of simulating deformations). In our implementation at that time we still compute it on the fly, by combining the Jacobian $J_{Uu}$ from barycentric to textural coordinates and the Jacobian $J_{Ux}$ from barycentric to surface coordinates that are both trivial to get ($J_{xu} = J_{Uu}.J_{Ux}^{-1}$). This only needs to be evaluated once per sprite. Moreover, we cache the result in order to avoid recomputing it if the next sprite lies on the same face of the mesh.

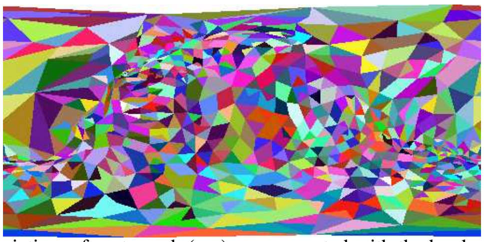

Fig. 9: The table associating a face to each (u,v), precomputed with the hardware using an item-buffer.

### 4.2.4 Finding a location

When the sprite shows an instant reaction to an instrument action, the location of this action (in particular the face) is known [1]. But during the simulation of the dynamics effects, especially for the rolling of blood drops on the surface, we need to know the location on the surface where a given $(u,v)$ texture location projects to, which corresponds to the reverse mapping. Finding quickly which face contains the given $(u,v)$ value is not easy. A 2D table indicating the subset of faces that intersect each given $[u,u+du]\times[v,v+dv]$ interval would decrease a lot the number of inclusion tests to be done, but it would still be costly to proceed the inclusion test, except if the table is large enough that each cell refers to a single face in almost every case. Once the face number is known, it is easy to compute the exact 3D location corresponding to the $(u,v)$ by linear interpolation of the face vertices. The solution we propose is very close to this, and the suggested table can be built using the hardware: we once render the organ in the texture space (see Fig. 9) using the $(u,v)$ instead of the $(x,y,z)$ as the vertices coordinates, and using for face color the face number (with the shading disabled) [2]. The resulting image is uploaded in main memory before the simulation, then used as a

[1] During our tests we use the mouse, getting the contact location using the feedback rendering mode (*picking*). In our integrated platform the tool is controlled in 3D using a Phantom© arm, and we rely on the (Lombardo et al. 1999) hardware accelerated collision detection technique to get this location.

[2] This technique is called the *item-buffer* and has been used previously many times, for instance in (Baum and Winget 1990) for the computation of form-factors using the hardware. It has been originally introduced by (Weghorst et al. 1984) to accelerate ray-tracing. To be noted that we use it in texture space, instead of geometric space.

table associating a face number to each $(u, v)$. Special care has to be taken for the faces crossing a singularity (a pole or the 'date change line') in order to draw them correctly, as explained on Fig. 10: each pixel should be covered exactly once.

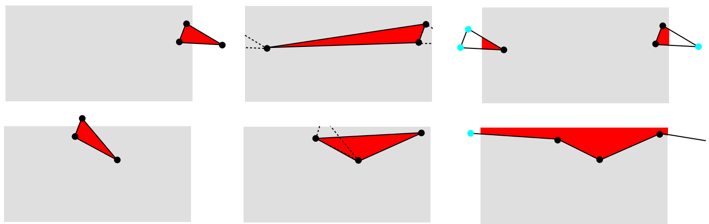

Fig. 10: Wrapping issues on the texture planisphere, at the date change line (top) and at the poles (bottom). We might want to draw a triangle that goes beyond the texture limits (left). The vertices are wrapped using a simple modulo, but they no longer describe the right triangle (middle). We have to clip correctly the triangles against the texture border (right).

#### 4.2.5 Optimizing the texture transfer

The only parts of the effects texture that have changed since the last time step and need to be updated on the graphics board are the areas covered by the active sprites plus the ones of the sprites that have just disappeared (i.e. the background should probably appear). The first idea is to transfer only the bounding box area of these sprites (in OpenGL it corresponds to the *subtexture* feature). But the sprites generally overlap, so that the total number of pixels transfered could be very large (see Fig. 11). The bounding box of all the sprites belonging to a given snake may also be the unit of transfer. But when the sprites lie on the diagonal of this box, a lot of useless pixels are transfered. The optimum is in between: we group the sprites by a given amount, which is estimated to balance the redundant pixels and the useless pixels.

If a snake contains $N$ sprites of size $L \times H$ that have a $(dx, dy)$ offset between each other, then if we make $k$ groups of sprites (assuming $N/K$ is integer), the number of pixels transfered[3] is $k((N/k-1)dx + L)((N/k-1)dy + H)$. The pixel saving between

[3] One group contains $n = N/k$ sprites. If these are evenly spaced, the group is $(n-1)dx + L$ large and $(n-1)dy + H$ high.

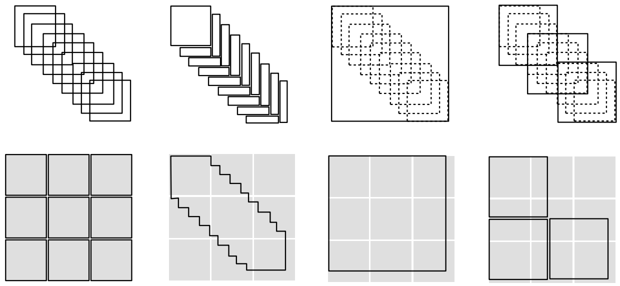

Fig. 11: Minimizing the pixel transfer when drawing snakes. *Left*: as the sprites overlap, transferring successively every sprites is redundant (on the bottom row, the amount of transfered data is figured). *Middle-left*: an optimum pixel set with no redundancy can be defined. Alas, the amount of context switches going with numerous blocks to be transfered and the limitations concerning memory alignment (disadvantaging small dimensions) make this solution non practical. *Middle-right*: transferring the whole snake's bounding box as one single block avoid pixels redundancy, but useless pixels are also transfered. *Right*: transferring the bounding box of groups of sprites is a tradeoff between the amount of redundant or useless pixels. An optimum can be found.

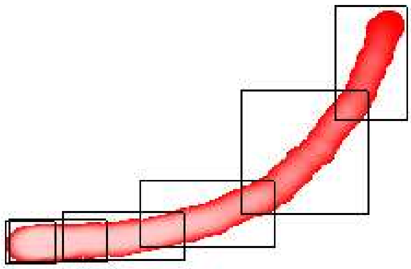

Fig. 12: Near-optimum texture area to transfer.

$k$ grouping and $(k+1)$ grouping is the difference $-N^2dxdy/k(k+1)+dxdy+LH-(dxH+dyL)$. This gain is positive as long as $\sqrt{k(k+1)} < N/\sqrt{ab-(a+b)+1}$ with $a = L/dx$ and $b = H/dy$. $\sqrt{k(k+1)}$ is close to $k$ for $k$ large enough. Our choice is to take for $k$ the integer part of $N/\sqrt{ab-(a+b)+1}$ (using the mean values for $L, H, dx$ and $dy$). An example of near-optimum grouping is presented on Fig. 12.

As stated above special care has to be taken when a sprite overlaps the 'date change line'. Such a sprite is split into two rectangles appearing on the two opposite sides of the map. For the optimum transfer evaluation, the parts of the snake that fall before

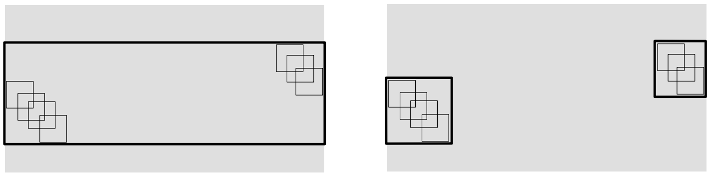

Fig. 13: When a snake is wrapped on the date change line, locality is lost and a bounding box contains a lot of useless pixels (left). Each wrapped portion of a snake should be considered independently (right).

and after the singularity are considered as two different snakes as figured on Fig. 13.

### 4.3 Reflection Map

On an organ the specularity is generally due to wetness, which yields mirror reflection on very wet locations and rougher reflection on the locations where the wetness layer is thiner, sticking to the surface grain. We could set an image of the light source in the reflection texture and simulate the roughness by accumulating rendering passes using slightly modified normals. This would obviously be costly. We precompute instead the effect of these by setting in the reflection texture (which constitutes the third superimposed layer) the convolution of the perfect mirror image of the light source (i.e. a ring) and a Gaussian 2D kernel figuring the normals variations (see Figs. 14 and 15). The first is scaled inversely with the light distance, while the second is scaled proportionally to the roughness (i.e. the Gaussian standard deviation). Thanks to the radial symmetry, we can even use an 1D radial Gaussian kernel [4].

Alas, this texture has to change with the distance of the light, and should thus be re-computed and transferred at each time step during a camera move, which certainly reduces the frame-rate. We introduce a solution that takes advantage of the spot symmetry: since all the points in the environment texture that are at the same angular distance of the optic axis (located at the center of the map) will show the same highlight intensity, we replace this explicit reflection map by the composition of a map of

[4] This convolution could be computed using the graphics hardware as done in (Soler and Sillion 1998) to get soft shadows.

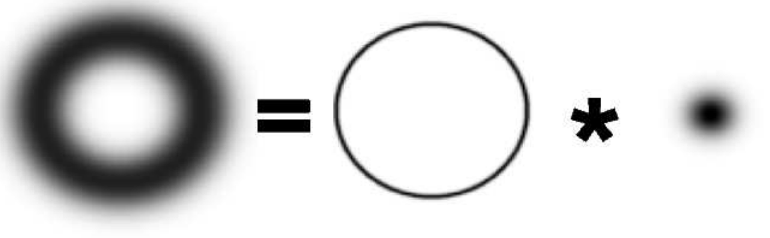

Fig. 14: The reflect texture is the convolution of the image of the source (i.e. a ring) and the roughness signature (i.e. a Gaussian).

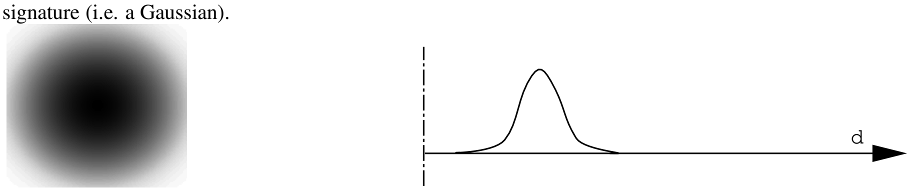


Fig. 15: The 2D radial ramp and the 1D reflect profile (i.e. the glossy ring seen in 1D).

the angular distances (i.e. a radial ramp) and an indirection table (i.e. a look-up table) that converts a radial distance into intensity. That way, the 2D map is constant and loaded once, while the 1D look-up table can easily be updated at every frame. It contains the convolution of the light ring (in 1D a simple peak) by a Gaussian, that is simply a translated Gaussian (as figured on Fig. 15). In fact, the 2D map can have a very low resolution, as the values will be recovered by the texture interpolation. The feature used for the indirection is the *texture color look-up table* of OpenGL, which is hardware-accelerated only on high-end graphics boards (such as the Infinite Reality). On low-end graphics boards, this can be emulated using color index textures: the look-up table is the color palette. To be noted that on the SGI O2 the textures are stored in main memory, thus suffering little penalty for texture loading.

**Highlight on drops**

In our implementation, we do not treat the highlights on spots in particular. If the reflect layer is drawn after the effect layer then the highlights cover the drops as well, in the opposite case the drops have no highlight. It would be easy to add a small white disk in the blood sprite in order to simulate an ideal drop highlight, but this would'nt correspond to reality as blood appears as flat spots rather than spherical drops. A better solution would be to apply the reflection map on the effect layer using a smaller roughness coefficient than for the skin. As this would require more passes, we preferred to neglect handling specific highlights for drops.

# 5 Evolving Components

## 5.1 Textural snakes of sprites

As we have suggested in section 4.2, we represent the various effects by a list of snakes, simulating three kinds of phenomena. A snake consists of a list of sprites. The motion of an effect is suggested by creating a new head and aging the previous sprites. An automaton manages the change in appearance as a sprite ages depending on its kind, and when a sprite should be deactivated. This happens when a sprite is at the tail of a drop, becomes too transparent, or gets the status of a permanent effect, as for a burnt spot. We store in each sprite structure its geometric characteristics (radius, bounding box), its automaton parameters (age, opacity), and a pointer to its location in the texture in main memory. The dynamics characteristics (velocity) only exist for the head, so we store them in the snake structure.

## 5.2 Automata

The snakes for drops are usually the longest. We design a blood sprite as a red disk with an opacity proportional to $1 - \frac{1}{(r/R)^2}$ where $R$ is the sprite radius and $r$ the distance of the current sprite pixel to the drop center. The aging consists of the multiplication of the global opacity by a fading coefficient. A sprite is deleted when its opacity is too weak or if a maximum length (i.e. age) is reached. In case of collision with another drop, which is tested using the sprite list associated with the face where the head sprite

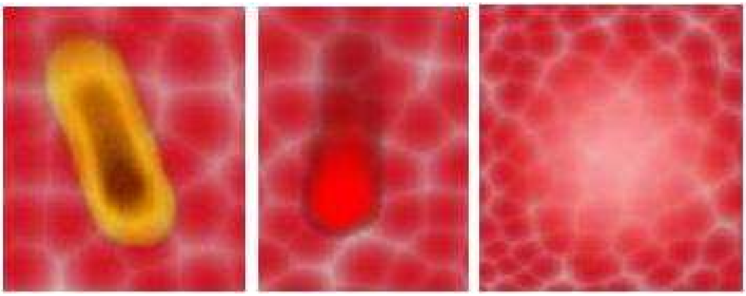

Fig. 16: The three kinds of dynamics effects: cauterization, blood drops and whitening.

lies, the tail of the collided drop is canceled and the radius of the collider increases.

The whitening corresponds to large, semi-transparent white disks. The head size and opacity increases as its cause of creation persists (i.e. contact of the tool with pressure), then the fading occurs by decreasing slowly the opacity and size.

The cauterization works in quite a different way: as the effect of heat depends on the previous state of the skin at this location (e.g. the physician might pass the cauterization tool several times over a location in order to cut the skin), a memory of this state needs to be maintained out of the snakes, otherwise we should keep the burning sprites forever just in case they might have to be reactivated. Thus we store an extra map in main memory containing the amount of heat received in each texture pixel, which is incremented at each new sprite creation. A software lookup table is then used to turn the heat received into color before drawing the sprite, from yellow for a weak burning (used by the physician for marking) to brown-black for cauterization. The snake is of minimal size for this kind as the sprites get a permanent effect immediately.

## 5.3 Drop motion

The drops have the most freedom: once created as a consequence of the user action, they evolve by themselves, under the influence of gravity. As for the other snakes, the head is the main element: the motion of the drop is simulated by creating a new sprite at the new head location and by fading the previous sprites. While for the other effects the new location of the head is simply the new site where the instrument acts on the organ skin, for a drop it results from Newtonian physics. The issue here lies in the coordination of the different spaces: forces are expressed in the 3D world, while the head sprite belongs to the texture world. To cope with this, we proceed as follow. The acceleration is computed in 3D, taking into account the gravity, the friction and the surface reaction. The evaluation of this last force requires the knowledge of the surface normal at that location. Thanks to the inverse mapping table (cf section 4.2.4), we know the face where the $(u, v)$ location of the sprite lies, which allow us to compute the normal by linear interpolation of the normals at the face vertices. The 3D velocity is then updated according to the acceleration and projected on the surface. The Jacobian of the mapping at this face is then used to convert this 3D velocity into a 2D displacement in the texture space.

Note that the friction is lower on a wet area: a drop coming there rolls faster. Such an area occurs where a drop has recently been, which means that the drop head is colliding with a drop tail that is there. We can simulate this phenomena thanks to the collision management. This is fortunate because this results in the merging of drops, which has the good effect of preventing the growth of the total amount of drops.

## 5.4 Real-time balance

If the rendering needs more than a $25^{th}$ of second, the simulation is no longer real-time. This has two consequences: the screen refresh rate is less comfortable, and the simulation time base no longer matches the user time base.

The last issue is the worst, and is corrected in the following way: we measure the delay $dt$ between the two previous frames, and we use this value as the physical time step in the simulation. This allows the simulation to run correctly (e.g. the distance covered by drops) whether the rendering is very quick or very slow.

Another problem occurs due to the events generated by the interface: the processing of an event includes a localization determination that is similar to a *picking* operation (this corresponds to the *feed-back buffer* feature on OpenGL). This requires a partial rendering pass with no effect on the screen, in order to determine which face is below the pointer (either 2D, e.g. the mouse, or 3D, e.g. the Phantom© arm). If too many events per second are generated by the computer, these extra renderings can saturate the pipe-line. We thus have to discard a proportion of events in such a way that this does not occur.

# 6 Results

We have implemented and tested the proposed solutions on an Onyx2 Infinite Reality2 and on SGI O2, as shown on Figs. 4, 6, 16 and 17. On the IR2, a regular frame rate of 25 fps is obtained while simulating 10 drops of 30 sprites, which allows further extensions. The frame rate is 8 fps on O2 in the same conditions. Without the moving drops, rendering the three layers can be done at more than 50 fps on the IR2 and 14 fps on the O2.

The liver geometric model is extracted from scanner data, and its surface is simplified down to 1200 triangles (the volumetric simulation of deformations cannot handle more in real time). The introduction of extra vertices to superimpose the regular texture mesh used for the skin layer brings the amount of triangles up to 3300, which is still very light for hardware rendering.

First demonstration to the physicians of our research project shows their deep interest and belief that this will lead to a usable training tool.

**Evolution of graphics accelerators**

New powerful graphics boards such as the Nvidia have appeared by the end of the project. It would be interesting to investigate what could be saved or extended using these boards. Multi-texturing and per-pixel shading would probably allow to simplify the rendering and decrease the number of passes. Bump mapping ability could certainly improve the skin aspect. On the other hand we rely on feedback rendering, subtextures and lookup tables, which are not yet handled by these new boards. Nevertheless, we hope to have proved that the amount and quality of visual effects that can be handled in a real-time simulation strongly depends on the richness of the features offered by the boards, more than the brute amount of polygons per second they can draw.

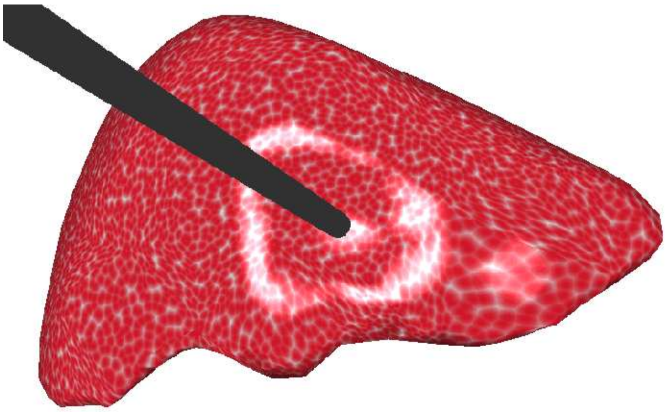

Fig. 17: A specular ring underlying the curvature change due to the contact of the instrument.

# 7 Conclusions

We have reached our aim of getting a real-time rendering for our laparoscopic simulator including the targeted features. The rendering stage has been integrated with the animation stage, as illustrated on Fig. 17.

Of course the work is far from being finished. We first need to tune all the colors and textures in coordination with physicians[5] in order to match the realism issue needed for the quality of immersion. Then we will have to address the simulation of our first surgery, the cutting and ablation of a part of the liver. This creates new surfaces whose surface area increases with time, yielding non trivial texturing problems. Vessels

[5]For instance, the skin texture used in this paper corresponds to a pig liver, which is the animal used for the training. It should better be human liver skin.

and ‘garbage’ should also appear between the two internal surfaces. Concerning the realism, surrounding organs should be introduced to help believing the simulation (real livers are not floating in the air). Moreover, these organs interact with the main organ. The grain of the liver skin appearing in the highlight could also be simulated. These are some of the goals of CAESARE, the cooperative project continuing AISIM.


### Acknowledgments

AISIM was a cooperative research project founded by INRIA from 1997 to 1999. It is continued by the CAESARE cooperative project founded by the French Research Department from 2000 to 2002. We thank Hervé Delingette who has initiated this project, and Jean-Christophe Lombardo who has animated it during the three years. We thank Pierre-Olivier Agliati, Antoine Leroy and Sylvain Trimoreau, who have worked during their training on the integration of the various rendering methods with the simulation platform. Thanks are also due to James Stewart who carefully reread this paper.

# A pseudo-algorithm

```
load liver mesh
load texture mesh
parameterize extra vertices with barycentric coordinates
load the four triangular skin patterns
send them to the graphics board
load the (u,v) of the regular nodes relatively to the
triangular tiles
build (u,v) for the effect map (spherical projection)
build the face(u,v) table using the hardware
build the radial ramp texture for the reflection map
send it to the graphics board

repeat forever:

/* rendering */
given the point of view, set the ModelView matrix
draw the skin layer:
bind triangle pattern 1
draw faces textured with it, using (u,v) relative to the tiles
same for pattern 2,3 and 4
draw the effects layer:
blending on
bind the effect texture
draw faces, using built (u,v)
draw the reflects layer:
blending on
set the spheremap (u,v) calculation mode
bind the reflect texture (i.e.  the radial ramp)
build the 1D reflect profile (knowing camera distance and
skin roughness)
send it to the graphics board (as texture lookup or
palette)
draw the faces
```

```
/* simulation */
handle user events, test for collisions with instruments
-> proceed liver distortion (out of the scope of this pa-
per)
-> new sprites are created
update effect texture:
erase sprites in effect texture in main memory
delete dead sprites
add created sprites
paint sprites in effect texture in main memory
send modified parts in effect texture on the graphics
board
execute sprites automata (color and transparency changes)
proceed motion:
for each moving sprite (i.e.  head of the moving snakes)
get face(u,v)
compute N
compute V from Newton laws
compute motion in texture space
update (u,v)
handle collisions
```

## References

[AISIM project ] AISIM project. Relevant URLs.
AISIM project: *(remark: the version in french is more recent)*
`http://www-sop.inria.fr/epidaure/AISIM/`
CAESARE project:
`http://www-sop.inria.fr/epidaure/CAESARE/`
Real-time simulation of deformations:
`http://www-imagis.imag.fr/~Marie-Paule.Cani/foie.html`
`http://www-imagis.imag.fr/~Gilles.Debunne/`
Real-time rendering:
`http://www-imagis.imag.fr/~Fabrice.Neyret/laparo/`.

[Baum and Winget 1990] Baum, D. R. and J. M. Winget. Real time radiosity through parallel processing and hardware acceleration. *Computer Graphics (1990 Symposium on Interactive 3D Graphics) 24*(2), 67–75.

[Catmull 1974] Catmull, E. E. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.d. thesis, University of Utah.

[Cotin et al. 2000] Cotin, S., H. Delingette, and N. Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. *The Visual Computer 16*(8), 437–452.

[Debunne et al. 1999] Debunne, G., M. Desbrun, A. Barr, and M.-P. Cani. Interactive multiresolution animation of deformable models. In *10th Eurographics Workshop on Computer Animation and Simulation (CAS'99)*.

[Debunne et al. 2000] Debunne, G., M. Desbrun, M.-P. Cani, and A. Barr. Adaptive simulation of soft bodies in real-time. In *Computer Animation 2000*.

[Fournier et al. 1998] Fournier, P., A. Habibi, and P. Poulin. Simulating the flow of liquid droplets. In *Graphics Interface'98*, pp. 133–142.

[Hanrahan and Haeberli 1990] Hanrahan, P. and P. E. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. In F. Baskett (Ed.), *Computer Graphics (SIGGRAPH '90 Proceedings)*, Volume 24, pp. 215–223.

[Lombardo et al. 1999] Lombardo, J.-C., M.-P. Cani, and F. Neyret. Real-time collision detection for virtual surgery. In *Computer Animation'99*.

[Miller and Hoffman 1984] Miller, G. S. and C. R. Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments. In *SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes*.

[Neider et al. 1993] Neider, J., T. Davis, and M. Woo. *OpenGL Programming Guide*. Reading MA: Addison-Wesley.

[Neyret and Cani 1999] Neyret, F. and M.-P. Cani. Pattern-based texturing revisited. In *SIGGRAPH 99 Conference Proceedings*, pp. 235–242. ACM SIGGRAPH: Addison Wesley.

[Nvidia ] Nvidia. Developer - white papers. `http://www.nvidia.com/Marketing/Developer/DevRel.nsf/WhitepapersFrame`.

[Perlin 1985] Perlin, K. An image synthesizer. In B. A. Barsky (Ed.), *Computer Graphics (SIGGRAPH '85 Proceedings)*, Volume 19(3), pp. 287–296.

[Picinbono et al. 2000] Picinbono, G., J.-C. Lombardo, H. Delingette, and N. Ayache. Anisotropic Elasticity and Forces Extrapolation to Improve Realism of Surgery Simulation. In *ICRA2000: IEEE International Conference Robotics and Automation*.

[Reeves 1983] Reeves, W. T. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics 2*, 91–108.

[Reeves and Blau 1985] Reeves, W. T. and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In B. A. Barsky (Ed.), *Computer Graphics (SIGGRAPH '85 Proceedings)*, Volume 19(3), pp. 313–322.

[Siggraph Course Notes 1998] Siggraph Course Notes. *Advanced Graphics Programming Techniques Using OpenGL*. Addison-Wesley. `http://www.sgi.com/software/opengl/advanced98/notes/`.

[Silicon Graphics a] Silicon Graphics. *Way cool, way fast OpenGL rendering techniques*. `http://reality.sgi.com/opengl/tips/`.

[Silicon Graphics b] Silicon Graphics. *Witches Brew: source + docs on Impressive Programming*. `http://toolbox.sgi.com/TasteOfDT/src/exampleCode/WitchesBrew/`.

[Soler and Sillion 1998] Soler, C. and F. X. Sillion. Fast calculation of soft shadow textures using convolution. In M. Cohen (Ed.), *SIGGRAPH 98 Conference Proceedings*, pp. 321–332. ACM SIGGRAPH: Addison Wesley.

[Weghorst et al. 1984] Weghorst, H., G. Hooper, and D. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics 3*(1), 52–69.

[Worley 1996] Worley, S. P. A cellular texturing basis function. In H. Rushmeier (Ed.), *SIGGRAPH 96 Conference Proceedings*, pp. 291–294. ACM SIGGRAPH: Addison Wesley.