GPU real time hatching

Suarez Jordane
Université Paris 8
2 rue de la liberté
93526, Saint Denis,
France
suarez@ai.univ-paris8.fr

Belhadj Farès
Université Paris 8
2 rue de la liberté
93526, Saint Denis,
France
amsi@ai.univ-paris8.fr

Boyer Vincent
Université Paris 8
2 rue de la liberté
93526, Saint Denis,
France
boyer@ai.univ-paris8.fr

ABSTRACT

Hatching is a shading technique in which tone is represented by a series of strokes or lines. Drawing using this technique should follow three criteria: the lighting, the object geometry and its material. These criteria respectively provide tone, geometric motif orientation and geometric motif style. We present a GPU real time approach of hatching strokes over arbitrary surfaces. Our method is based on a coherent and consistent model texture mapping and takes into account these three criteria. The triangle adjacency primitive is used to provide a coherent stylization over the model. Our model computes hatching parameter per fragment according to the light direction and the geometry and generates hatching rendering taking into account these parameters and a lighting model. Dedicated textures can easily be created off-line to depict material properties for any kind of object. As our GPU model is designed to deal with texture resolutions, consistent mapping and geometry in the object space, it provides real time rendering while avoiding popping and shower-door effects.

Keywords

non-photorealistic rendering, hatching, stroke based texture, shading, GPU

1 INTRODUCTION

Hatching is an artistic drawing technique that consists in drawing closely spaced lines to represent objects. Artists depict tonal or shading effects with short lines. Parallel lines or cross lines can be used to produce respectively hatching or cross-hatching. Hatching is commonly used in artistic drawing such as comics but is also largely used in specific visualization systems such as archeology, industry and anatomy. Hatching can also be used as a common way to promote any kind of products. Hatching generally refers only to lines or strokes. We prefer the term of geometric motifs including various possible patterns and dots, lines or cross lines. Artists can thus provide hatching, cross-hatching as well as stippling.

Hatching is produced by the artists following three criteria: lighting, object geometry and object material. These criteria provide tone, geometric motif orientation and geometric motif style. The tone of the geometric motif used for hatching refers to the lighting equation. Geometric motif orientation is provided by the object geometry and the direction of light. Finally, depending on the object material, different geometric motifs can be used.

This paper presents a method for real time hatching considering these three criteria. Our model is relevant in all interactive applications such as games, scientific, educational or technical illustrations. As it is often mentioned in non-photorealistic rendering (NPR) work, the lack of temporal coherence in stroke placements and image-based approaches respectively produce flickering and shower-door effect during animations.

We deal with objec-space coherence by considering that stroke width and density should be adjusted to camera, lighting and scene transformation. Desired tones should be maintained even as the object moves toward or away from the camera. We propose a full GPU implementation able to generate hatching strokes on 3D scene. As [Pra01], we address the same challenges: (1) limited run-time computations, (2) frame-to-frame coherence among strokes, and (3) control over stroke size and density under dynamic viewing conditions.

But by opposition to the previous work, we consider that the stroke orientations directly depend on the light direction and should be computed per fragment. In

fact, as mentioned above, artists choose the geometric motif orientation according to the geometry and the light direction. As artists do, we provide a model able to consider these two constraints. Rare are the previous work in which the light direction is considered. In our approach, we address these challenges by providing a new texture mapping model that deals with modern geometry stage hardware.

In this paper, we provide a hatching rendering that satisfies the three criteria. This is achieved noticeably through:

- Hatching parameter computations per fragment according to the light direction and the triangles topology;
- Generation of hatching renderings that take into account these parameters and a lighting model.

We first present the related work in NPR domain dedicated to stroke based renderings. Then we present our model and its implementation. The performance of our model are discussed and the results demonstrate the relevance of our solution in the hatching problem. Finally, we conclude and propose future work.

2 PREVIOUS WORK

Hatching can be considered as a subset of drawing or painting style in which individual strokes are produced. Most related work have been presented as NPR papers. For that reason, we present hereafter related work on stroke based NPR.
Stroke based NPR can be categorized into two kinds of methods depending on the type of treated primitives. Some focus on pixel buffers while others deal with geometric objects. Below, we present general stroke based methods according to this classification, image space or object space, and particularly focus on specific hatching ones.

Image space methods use images in order to extract color, depth or tone information. In this kind of approaches the main challenge lies in geometric motif orientation:
In [Hae90], P. Haeberli has presented a method that creates impressionist paintings from images using "brush-strokes". The orientation of strokes depends on image gradient and direction fields.
In [Sal94], the proposed method aims to generate pen and ink illustrations from images by placing stroke textures. The orientation of these textures is computed according to the image gradient and the user defined directions.
In [Win96], the authors have created pen and ink rendering from a viewpoint of 3D scenes. To achieve this goal, the method uses controlled density hatching and user defined directions.
The model proposed in [Sal97] produces pen and ink style lines from images or a viewpoint of 3D models. In this model, strokes are oriented according to the direction fields.
In [Sai90], the authors have proposed to produce line drawing illustration from a viewpoint of 3D scenes according to curved hatchings and "G-Buffer". This specific geometric buffer contains geometric surface properties such as the depth or the normal vector of each pixel. The curvature is extracted for each object and indicates its topology. Hatching is then automatically produced using curvature and other information like rotations or texture coordinates.

The main drawback of these methods remains the appearance of the shower-door effect. In fact, as the geometry of these scenes is rarely considered, lighting, depth and other 3D information are almost never taken into account.
Then, we prefer approaches that focus on 3D model information.

In object space methods, additional information such as light position, depth and normal of each vertex can be extracted from geometry:
In [Deu00], the proposed method generates pen and ink illustrations using different styles such as the cross hatching and according to the object normals and the scene viewpoint.
The method presented in [Kap00] creates some artistic renderings particularly "geograftals" strokes (oriented geometric textures) according to the principal curvature of the 3D models.
In [Lak00], Lake & al. have presented a method that produces stylized renderings like pencil and stroke sketch shading which orientation depends method also allows the user to create cartoon rendering of 3D models.
In [Her00], the authors have proposed a method to generate line art renderings using hatch mark and generated smooth direction fields.
In [Pra01], non-photorealistic renderings with textures of hatching strokes called "tonal art map"(TAM) are described. Multitexturing is used for rendering TAMs with spatial and temporal coherence. Orientation of textures is computed according to a curvature based on direction fields and lapped texture parametrization (set of parametrized patches).
In [Web02], Webb & al. have presented two real time hatching schemes based on TAM. This method avoids blending or aliasing artifacts using a finer TAM resolution. We can notice that one of this scheme supports colored hatching.

In [Coc06], a model for pen and ink illustrations using real time hatching shadings is proposed. This approach is hybrid and combines image space and object space stroke orientation methods. Stroke orientations are computed according to a combination of dynamic object space parameterizations and view dependent ones. It permits a better spatial coherence than the object-space approaches and reduces the shower-door effect compared to image-based hatching methods but is dedicated to specific geometries such as tree models.

The main drawback of this kind of methods is the texture discontinuity. As texture coordinates are independently generated for each triangle, a possible discontinuity may be introduced and visible artifacts may be generated.

3 OUR MODEL

We aim to produce hatchings according to the previously described three criteria. Tone, form and material should provide a coherent and consistent hatching. As described in [Pra01], we think that a texture based approach is suitable to achieve the hatching and different materials can be depicted by the texture variety. Form and tone should be the result of a lighting model but by opposition to the related work, in our approach, the geometric motif (i.e strokes or lines for example) orientation also depends on the light direction and not only on the model curvature. As a consequence, textures should contain set of continuous tones.
We detail our solution realized in four steps:

1. The generation of textures representing tones: each generated texture represents a tone composed by a geometric motif (for example a set of lines, strokes or points). The set of generated textures should contain a set of continuous tones (see figure 1). Note that to compute a tone per fragment, we should determine a texture number and texture coordinates.

2. Texture orientation and texture coordinates generation: according to the light position and for each triangle, textures should be oriented in order to follow the light displacement. Texture coordinates generation must guarantee no texture distortion for the three vertices of each triangle. At this step, texture coordinates are computed for a given primitive and for each vertex (see section 3.2 and figure 3). This provides the first part of the paper contribution.

3. Texture continuity: we ensure the texture continuity between each triangle and its neighbors by computing a blend factor per neighbor (see section 3.3 and figure 8). This step permits a full computation taking into account each triangle and its neighbors and constitutes the second part of this paper contribution.

4. Tonal mapping: we compute the tone per fragment by interpolating vertex texture coordinates (step 2) and a light equation determining the texture number. This last computation is realized per fragment since the texture number is not obtained by interpolation (see figure 9). Finally, for a triangle, we blend results provided by neighbors according to the blend factor computed at step 3. Note that for a given fragment in a triangle, blend factors are interpolated (see figure 8).
   Moreover, even if we present our model with a set of textures, in practice, we have a multi-resolution set of textures providing mipmapping and avoiding aliasing (see figure 2). This last step generates the final rendering using previously computed values, a lighting model and a multi-resolution tonal art map.

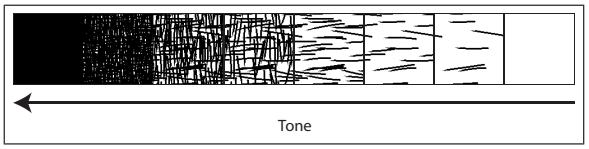

Figure 1: Tonal art map example composed by 8 different textures.

3.1 Texture generation

In our method, we use precomputed tonal art map introduced in [Pra01], where hatch strokes are mipmapped images corresponding to different tones.
Tonal art map must satisfy some consistency constraints:

- For a given tone, all textures must have the same grayscale average;
- geometric motifs present in a given texture must be present in all higher texture resolutions and superior tones;
- geometric motifs have the same form regardless to the texture resolution. For example, lines must have the same width and length and points must have the same radius.

As an example, figure 1 shows a set of continuous tones given at a high resolution (TAM). Figure 2 presents an example of multi-resolution tonal art map: a multi-resolution tone texture deduced from figure 1. Note that, as all mipmapping techniques, the multi-resolution TAM will be used to avoid aliasing by taking into account the fragment depth. So, as it is a well-known solution, we do not consider this problem hereafter and use only figure 1 throughout the article to illustrate our purpose. Once we have this texture palette, we want to determine the texture coordinates of each vertex of our 3D model.

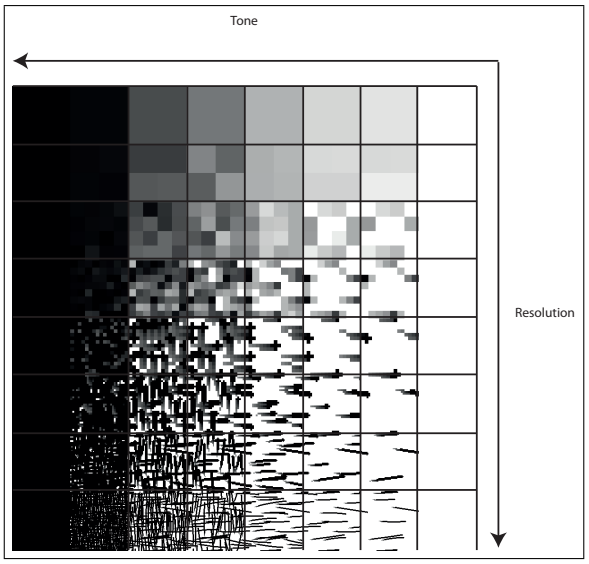

Figure 2: Multi-resolution tonal art map example.

3.2 Hatching texture orientation

As described above, our approach matches the geometric motif orientation depending on the light direction. Thus, as shown on figure 3 considering a given light source and a 3D model, we have to orient and crop the TAM according to the light source properties (position, direction, cut-off and spot exponent).

To describe our model, we use the following nomenclature. L describes a light, $\overrightarrow{L_d}$ depicts its direction while L_p depicts its position. Note that in the case of directional light, L_p is not used. T_c is a triangle of the 3D model and V_i ($i \in \{0,1,2\}$) are its vertices. Each vertex V_i, represented by its 3D space coordinates (x,y,z), is associated to a hatching parameter corresponding to its 2D texture coordinates noted $H_i^c(s,t)$. Figure 5 illustrates this notation used hereafter in the paper. As mentioned in the introduction of our model description, in this section we present a hatching parameter computation considering only one triangle (*i.e* generalization considering triangle and neighbors is detailed in the next section).

Since we want to guarantee no texture distortion in the considered triangle, the applied transformations should necessary be realized per primitive. This is done by computing, for each triangle, its hatching parameters according to the tangent space of the triangle itself. This tangent space is the Tangent-Binormal-Normal $(\overrightarrow{T},\overrightarrow{B},\overrightarrow{N})$ coordinates system with $\overrightarrow{B}$ representing the vector from the triangle toward the light, projected on the triangle plane. In the case of positional light with its position (L_p) and direction ($\overrightarrow{L_d}$), we compute the direction to the light per triangle ($\overrightarrow{L}$) considering G_c the triangle barycenter (*i.e* $\overrightarrow{L} = \overrightarrow{L_p - G_c}$). In the case of directional light, the direction to the light ($\overrightarrow{L}$) is equal to the opposite of the light direction ($\overrightarrow{L_d}$) (*i.e* $\overrightarrow{L} = -\overrightarrow{L_d}$). Finally, $\overrightarrow{B}$ is the normalized projected vector $\overrightarrow{L}$ in the triangle plane.

Then, for each vertex V_i of each triangle T_c, by applying a change of basis from the object local space to $(\overrightarrow{T},\overrightarrow{B},\overrightarrow{N})$ we obtain, for the corresponding $\overrightarrow{T}$ and $\overrightarrow{B}$ coordinates, the vertex 2D texture coordinates stored in H_i^c as shown in algorithm. 1:

$\overrightarrow{N} \leftarrow$ normalize $((\overrightarrow{V_1 - V_0}) \times (\overrightarrow{V_2 - V_0}))$
if *the light source is positional* **then**
  $G \leftarrow (V_0 + V_1 + V_2)/3$;
  $\overrightarrow{L} \leftarrow \overrightarrow{L_p - G}$;
else
  $\overrightarrow{L} \leftarrow -\overrightarrow{L_d}$;
end
$\overrightarrow{B} \leftarrow$ normalize$(\overrightarrow{L} - \overrightarrow{N} \cdot (\overrightarrow{L} \cdot \overrightarrow{N}))$;
$\overrightarrow{T} \leftarrow \overrightarrow{B} \times \overrightarrow{N}$;
foreach *i in* $\{0,1,2\}$ **do**
  $H_i^c.s \leftarrow V_i \cdot \overrightarrow{T}$;
  $H_i^c.t \leftarrow V_i \cdot \overrightarrow{B}$;
end

Algorithm 1: Computing hatching texture orientation per triangle.

This step is performed in the geometry shader stage, for each triangle T_c expressed in the local object space. Thus, we compute and emit for each V_i its position and texture coordinates.

We obtain, as shown in figure 4-(a), hatchings that take into account triangle orientations. The four triangles presented on this figure have different normals. Remark that hatchings on the common edges between triangles produce stroke discontinuities. These discontinuities are naturally due to the difference of orientation ($\overrightarrow{T}$) computed at this level but also to the difference of shading (which determines tones of the art map) computed at the last level (fragment shader stage). To avoid this discontinuity we propose to use the triangle adjacency information to compute multi-texturing coordinates and blend the result as shown on figure 4-(b). This process is explained hereafter.

3.3 Adjacency blending

In order to ensure a continuity of textures between two neighboring triangles we determine the contribution of adjacent triangles in the rendering of T_c.

As shown in figure 5, triangle adjacency is a geometric primitive composed by six vertices V_i ($i \in \{0,1,2,\alpha,\beta,\gamma\}$) describing four triangles where T_c is the current processed triangle and T_α, T_β and T_γ its adjacents. This primitive is accessible in the geometry shader stage where data of adjacent triangles (as vertex position) are accessible during T_c processing but are not emitted during this same process.

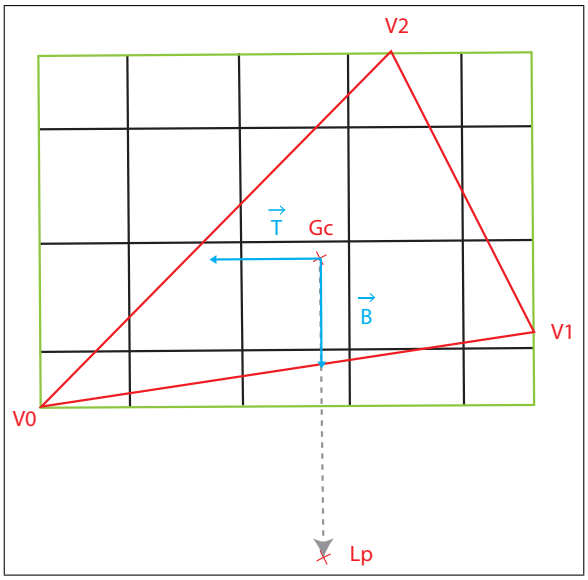

Figure 3: Texture orientation on a given triangle according to the projection of the light position.

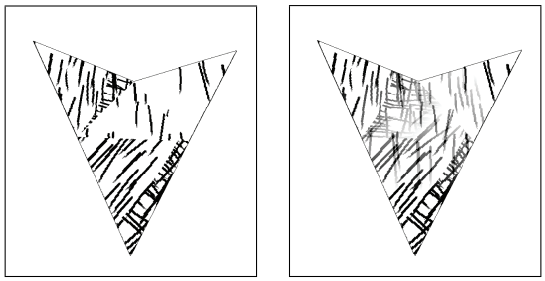
(a) (b)

Figure 4: Hatching results on 4 adjacent triangles with different normal vectors. (a): without blend, (b): with blend.

We keep the main calculations of subsection 3.2 by integrating data adjacency, calculating four different texture coordinates per vertex of T_c. In fact, for each vertex V_i ($i \in \{0,1,2\}$) we need to compute the hatching parameters corresponding to each triangle of the adjacency primitive. Thus, we obtain a 3D component H_i^t with $t \in \{c,\alpha,\beta,\gamma\}$ composed by 2D texture coordinates (s,t) and a blend factor f. H_i^t depicts, for the vertex V_i, the contribution of the triangle t in the current triangle hatching. Thus, the triangle is textured according to the orientation of its neighbors to the light. These coordinates are used to mix the results by blending. For each adjacent triangle, we compute its corresponding $(\overrightarrow{T},\overrightarrow{B},\overrightarrow{N})$ coordinates system which we use to compute the corresponding hatching parameters.

Figure 6 illustrates the calculation of hatching parameters where T_c is a triangle being processed and T_γ an adjacent triangle. In this case, $\overrightarrow{B}$ is given by $\overrightarrow{G_\gamma L_p}$ and $\overrightarrow{T}$ is calculated according to the N_γ (*i.e* the normal of T_γ) and $\overrightarrow{B}$.

V_1, the opposite vertex to the adjacent side is projected in the plane of T_γ. We obtain $V_1^{'}$ and use it to compute texture coordinates of the hatching parameters H_1^γ (hatching parameters of V_1 for T_γ). This projection is applied to all vertices of T_c and all triangles T_α, T_β and T_γ.

As a first approach, we propose to blend the results given by these different texture coordinates producing a rendering whose aspect is continuous at the junction of triangles. Thus, we calculate a blending factor f for each vertex per adjacent triangle. This first approach is a compromise producing grayscale strokes (see on figure 4(b)). In practice, these artifacts are hardly visible on a 3D model as demonstrated in the figure 10 and in the addtional video (i.e the results section). Note that complete approach is palnned in future work.

As shown in figures 7 and 8, for each vertex V_i, we compute a contribution value that gradually decreases along the triangle. Depending on the dot product between two adjacency triangle normals we ensure that when two triangles have an angle less than $\frac{\pi}{2}$ radian, there is no contribution between them. For example in a cube, triangles of different faces should not influence each other. So, finally we obtain four hatching parameters per vertex of T_c.

We present below the algorithm 2 in which we compute for each emitted vertex V_i (*i.e* $i \in \{0,1,2\}$) of T_c its direction to the light $\overrightarrow{V_iL}$, the triangle normals $\overrightarrow{N_t}$ (*i.e* $t \in \{c,\alpha,\beta,\gamma\}$) and its hatching parameters H_i^t. These values will be used in the final step to render the model according to a lighting model.

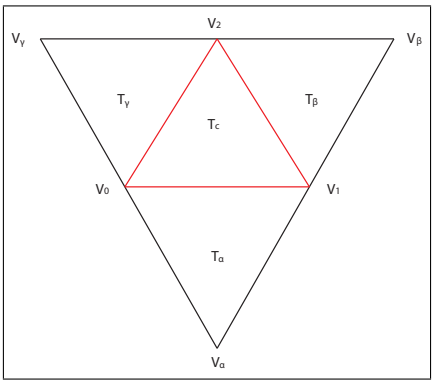

Figure 5: Triangle adjacency topology. The main triangle is T_c. Neighbor ones are indexed α, β, γ.

3.4 Tonal mapping

Our model is inspired both by the toon shading technique introduced in [Lak00] where the light intensity is given by a 1D texture and a Phong shading where contribution is computed per fragment.

This approach can be extended to 2D textures combining lighting to a palette of textures. In the fragment shader stage, by computing the dot product between fragment-light vector $\overrightarrow{FL}$ and the fragment nor-

foreach *triangle t in* $\{c, \alpha, \beta, \gamma\}$ **do**
  Compute normal and record it in $\overrightarrow{N_t}$;
  Compute projected light direction in T_t plane as previously described;
end
foreach *vertex i in* $\{0,1,2\}$ **do**
  foreach *triangle t in* $\{c, \alpha, \beta, \gamma\}$ **do**
    Compute V_i-light vector and record it in $\overrightarrow{V_iL}$;
    if $(t = c)$ **then**
      Compute texture coordinates of V_i as described in the previous subsection and record them in $H_i^c.st$;
      $H_i^c.f \leftarrow 1.0$;
      full contribution of T_c;
    else if V_i *does not belong to* T_t **then**
      $V_i^{'}$ is V_i projected in the T_t plane;
      Compute texture coordinates of $V_i^{'}$ in T_t and record them in $H_i^t.st$;
      $H_i^t.f \leftarrow -1.0$;
      no contribution of T_t;
    else
      Compute texture coordinates of V_i and record them in $H_i^t.st$;
      $H_i^t.f \leftarrow \overrightarrow{N_c} \cdot \overrightarrow{N_t}$;
      contribution depending on angle between T_c *and* T_t;
    end
  end
end

Algorithm 2: Compute Hatching parameters per Vertex according to adjacent triangles.

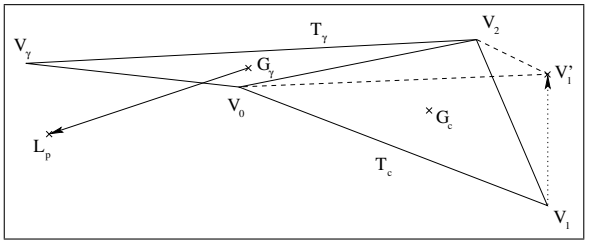

Figure 6: Texture coordinates obtained by adjacent triangle: used light direction and opposite vertex projection computation for continuity.

mal $\overrightarrow{N_t}$, we obtain, as shown in figure 9 the Lambertian term used to find the texture number in the TAM named *Tone*. Then, fragment information is automatically provided by linear interpolations in the GPU and according to each vertex information. Depending on the blending values $H_t.f$, the texture coordinates $H_t.st$, and the texture numbers *Tone*, we can compute, per fragment, the final color following the algorithm 3.

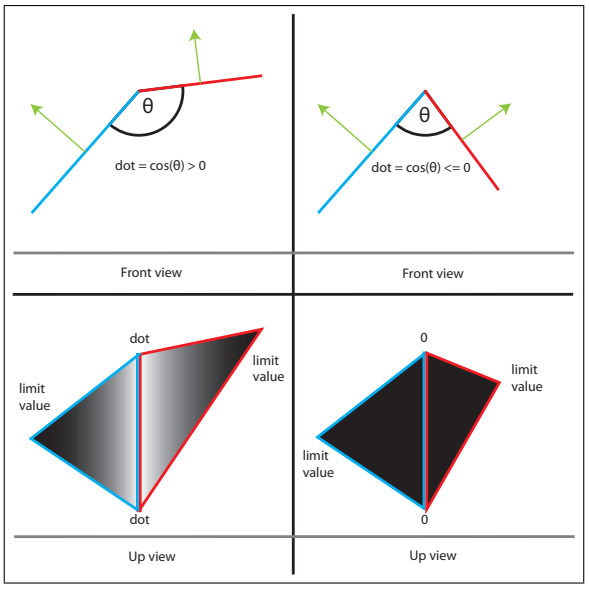

Figure 7: The blending value according to the angle between two adjacency triangles. Top of figure presents two front views of two triangles. On the left we study the case of $\cos\theta > 0$ and on the right we have the opposite case. Bottom of figure presents the blending factor computed per vertex and its interpolation along each triangle.

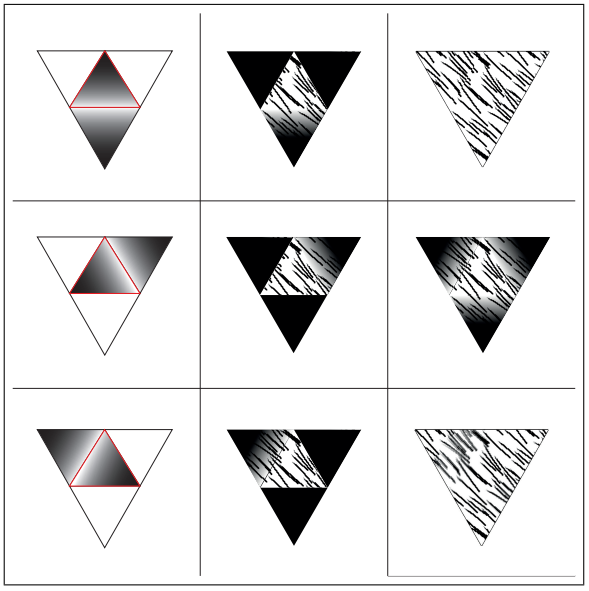

Figure 8: First column: contribution value between adjacent triangles. Second column: contribution applied to textures. Last column: result without blending, result of blending in T_c, result of blending in all triangles.

Note that, in our implementation, the level in the multiresolution TAM is chosen according to the fragment depth.

4 RESULTS

We present different results obtained with our model and discuss about performance, temporal and spatial

foreach *triangle t in* $\{c, \alpha, \beta, \gamma\}$ **do**
    Compute *Tone* according to $\overrightarrow{N_t}$ and $\overrightarrow{FL}$;
    Compute $Color_t$ determined by $H_t.st$ and *Tone*;
end
$FinalColor \leftarrow \sum(Color_t \times H_t.f) / \sum(H_t.f)$;
Algorithm 3: Compute the fragment color

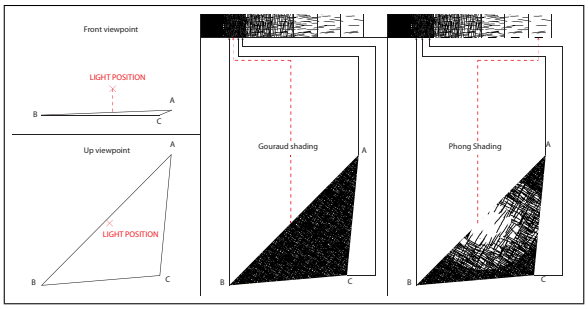

Figure 9: Example of texture number determining per fragment with different lighting techniques

coherence during scene animations.

Figure 10 presents an overview of possible results using our hatching model. The hatching parameters H_i^t of each vertex V_i can be globally modified on the fly, according to a matrix that provides the three basic transformations. Indeed, considering the $(\overrightarrow{T}, \overrightarrow{B}, \overrightarrow{N})$ coordinates system, we can shift $H_i^t(s,t)$ by adding a value in $[0;1]$ corresponding to a translation on $\overrightarrow{T}$ and/or $\overrightarrow{B}$ axis. We can scale each $H_i^t(s,t)$ using any scale value that modifies the texture repetition (see figure 10 second line). We can also modify $H_i^t(s,t)$ by making a rotation on the $\overrightarrow{N}$ axis to change the global texture orientation (see figure 10 first line and figure 14).

Our model provides a coherent lighting that follows the light and, as one can see on figure 11, reflects the fineness of the mesh.
Otherwise, our model gives the ability to represent different materials using different TAMs (see figure 12). TAMs can be used in addition to color materials (see figure 13 and figure 10).

Considering the performance aspect, our model is real time even for detailed geometry. Figure 15 illustrates our implementation performance expressed in number of frame per second considering different models. As one can see, for a 3D object composed by more than 500 000 triangles, our model produces renderings in 30 frames per second (NVIDIA Quadro FX 3800). As a comparison, we provide results both for our hatching model and for a basic fragment lighting model: a Phong shading. Note that, for both of them, we send the same geometry to the GPU including adjacency data. We obtain a ratio around 50% between these two renderings:

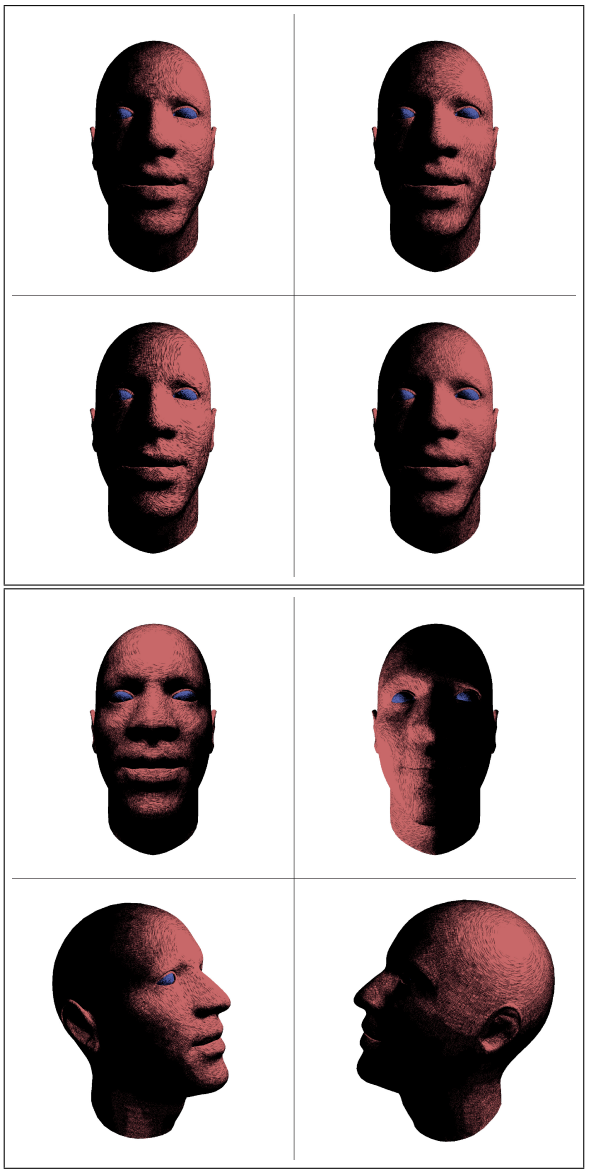
Figure 10: Results on 3D face model showing all different effects. First line: texture rotation along N axis for each triangle. Second line: different scale values are used. Third line: renderings using different light positions. Last line: model rotation for a fixed light.

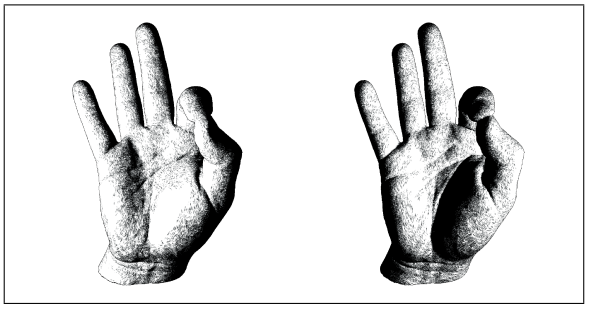
Figure 11: Results on the hand model with different light positions. Note that geometric details are always visible.

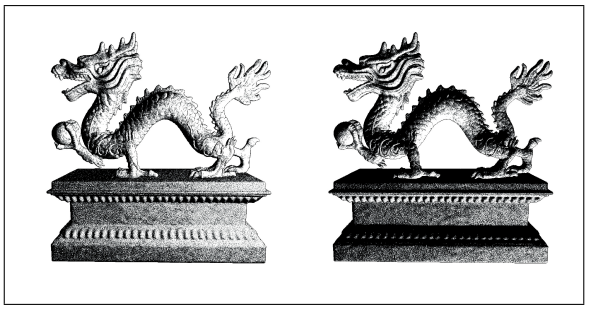

Figure 12: Results on the Stanford dragon model with different light positions and TAMs representing different materials.

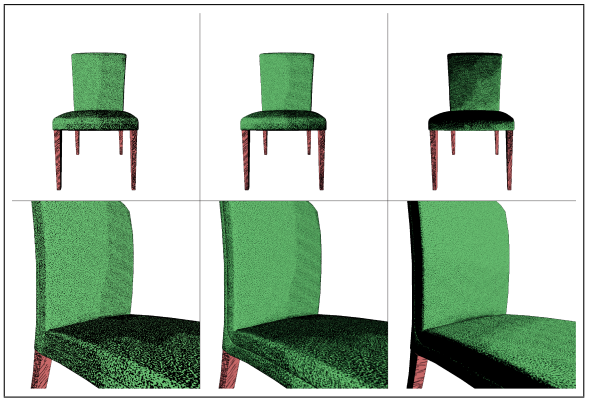

Figure 13: Results on a chair model with different light positions and TAMs + colorization representing distinct materials. First column shows results without adjacency blending.

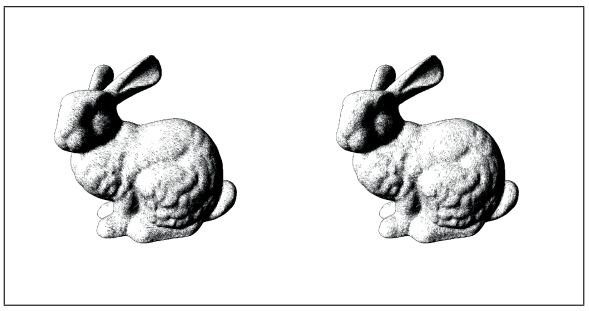

Figure 14: Results on the Stanford bunny model with different texture orientations. On the left, strokes are oriented toward the light. On the right, strokes are tangent to the light direction.

for a given geometry, our hatching rendering is twice slower than the Phong shading.

Concerning the spatial and temporal coherence of our model, a video showing our real time results is available at the following url:
`http://www.ai.univ-paris8.fr/~suarez`
We can notice that, between two consecutive frames, when lighting changes, variation of the selected TAM remains progressive while highlighting the object geometry.

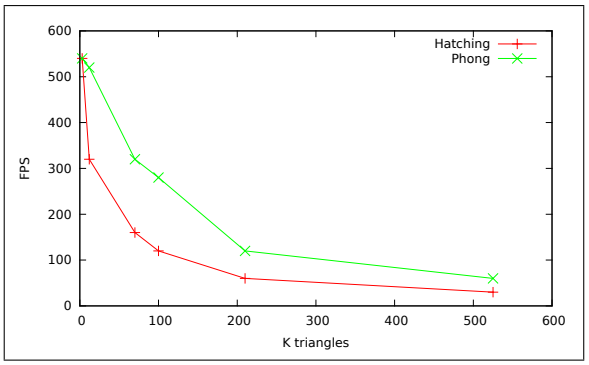

Figure 15: Graph showing performance of our rendering pipeline on different 3D models at a 1024x1024 rendering resolution (the used GPU is a Nvidia Quadro FX 3800). Results are expressed in FPS according to the number of triangles.

5 CONCLUSION

We have presented a model to produce hatching in 3D scene taking into account the brightness due to lighting, the orientation linked to the object geometry and the materials related to its texture. Our implementation is fully GPU and provides real time hatching on large scenes. Our model can be applied to any 3D models where the topology is constant. It provides hatching on static 3D models, animated 3D models and supports deformations. Triangle adjacency can be easily deduced from any 3D models at the loading step by indexing the model vertices. Moreover, no modeling engine modifications are needed. The model is also spacially and temporaly coherent since it gives continuous hatching during object animations and/or light displacements avoiding popping effects.

As future work, it will be interesting to have the ability to produce C_1-continuous strokes through adjacent faces. A procedural generation of TAMs constitutes an interesting way to adress this kind of problem. Moreover, the continuous aspect will be obtained automatically and dynamically without grayscale strokes. We also aim to manage multiple light sources in a single rendering pass by choosing a way that changes the hatching orientations (not simply blend them) according to these multiple sources. Finally, we plan to integrate drop and/or soft shadows and self-shadowing calculations to the model and then produce hatchings by disrupting orientations of faces affected by such shades.

6 REFERENCES

[Coc06] L. Coconu, O. Deussen, and H.-C. Hege. "Real-time pen-and-ink illustration of landscapes". In: *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pp. 27–35, ACM, New York, NY, USA, 2006.

[Deu00] O. Deussen and T. Strothotte. “Computer-generated pen-and-ink illustration of trees”. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 13–18, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[Hae90] P. Haeberli. “Paint by numbers: abstract image representations”. In: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pp. 207–214, ACM, New York, NY, USA, 1990.

[Her00] A. Hertzmann and D. Zorin. “Illustrating smooth surfaces”. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 517–526, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[Kap00] M. Kaplan, B. Gooch, and E. Cohen. “Interactive artistic rendering”. In: *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pp. 67–74, ACM, New York, NY, USA, 2000.

[Lak00] A. Lake, C. Marshall, M. Harris, and M. Blackstein. “Stylized rendering techniques for scalable real-time 3D animation”. In: *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pp. 13–20, ACM, New York, NY, USA, 2000.

[Pra01] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. “Real-time hatching”. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 581–586, ACM, New York, NY, USA, 2001.

[Sai90] T. Saito and T. Takahashi. “Comprehensible rendering of 3-D shapes”. In: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pp. 197–206, ACM, New York, NY, USA, 1990.

[Sal94] M. P. Salisbury, S. E. Anderson, R. Barzel, and D. H. Salesin. “Interactive pen-and-ink illustration”. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 101–108, ACM, New York, NY, USA, 1994.

[Sal97] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin. “Orientable textures for image-based pen-and-ink illustration”. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 401–406, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997.

[Web02] M. Webb, E. Praun, A. Finkelstein, and H. Hoppe. “Fine tone control in hardware hatching”. In: *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pp. 53–59, ACM, New York, NY, USA, 2002.

[Win96] G. Winkenbach and D. H. Salesin. “Rendering parametric surfaces in pen and ink”. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 469–476, ACM, New York, NY, USA, 1996.