Rendering Pen-drawings of 3D scenes on networked processors

Johan Claes, Patrick Monsieurs, Frank Van Reeth and Eddy Flerackers
Limburgs Universitair Centrum

Expertise centre for Digital Media
Wetenschapspark 2
B3590 Diepenbeek

Abstract
The subject of the paper is the computer-based rendering of bi-tonal images of 3D scenes looking alike pen-and-ink drawings generated manually. Traditional pen-drawings consist of thousands of pen-strokes, implying their generation can be a time consuming process. Especially when animated sequences of this kind of drawings are to be generated, manual techniques can become really time-consuming and cumbersome. Hence, computer-generated renderings of these types of images are well-worth investigating.

The paper refers to related work in this field and explains the various principles and techniques utilized in our work on this topic. Various aspects of tone and texture on the pen-and-ink drawings as well as the relevant aspects of the bi-tonal 'shading'-patterns are described. The various algorithms of the hybrid 2D/3D rendering process are given and commented. The realization of a networked rendering process -for single images as well as for animated sequences- is discussed. The influence of various parameters on the rendering process (resolution, use of compression, number of processors involved in the calculations) are elucidated. Results, conclusions and directions of future research are given.

1 Introduction

1.1 General

The generation of pen-and-ink drawings manually is an art-form and an illustration technique that has earned its merits in many fields such as architecture [Klitment 84], graphical design, advertising, etc. Manual composition and creation of pen-drawings, often consisting of many thousands of pen-strokes, inherently is a time consuming process. When the pen-and-ink illustrations are composed of many textured faces, the image generation takes even more effort. The technique becomes really cumbersome and time-consuming in cases where animated sequences of this kind of drawings have to be made. Hence, generating these types of images with the aid of computers is a solution worth investigating.

1.2 Related Work

References to related work can be divided in various categories, relating to work around (i) pen-and-ink illustrations, (ii) rendering of brush/pen strokes, (iii) computer generation of animated sequences looking alike traditional animation, and (iv) distributed rendering techniques:

(i) In the literature, various efforts have been described that realize computer-generated pen-and-ink illustrations [Salisbury et. al. 94], [Winkenbach et. al. 94]. Although the overall objective in these works is the same as in our effort, some definitive differences in realizing the objective can be indicated. In [Salisbury et. al. 94] the emphasis is put on the interactive guidance of the illustrations, whereas the main differences with [Winkenbach et. al. 94] is

found in the clipping technique applied in the rendering algorithm (Boolean operators on projected polygons (cf. Section 2.3) versus BSP tree approach) as well as in the determination of the shading ('statistical' determination of shading (cf. Section 2.4) versus incremental shading).

(ii) The definition and rendering of brushes in general has received considerable attention in the literature [Haeberli 90], [Cockshott et. al. 92], [Strassman 86]. More in particular, the generation of pen strokes [Hsu et. al. 94] and the utilization of curves as a basis for this [Finkelstein et. al. 94] has been reported upon as well. The approach we opted for in the definition/generation of strokes is elucidated in Section 2.4.

(iii) In the efforts covering the generation animated sequences with a likeness of that encountered in traditional animation [Litwinowicz 91], [Van Reeth 96], it is the intention to utilize tools and techniques that support the animation production for speeding it up or for increasing the functionality. Once the generation of pen-and-ink illustration of single images by means of computers is a fact, the extension towards the generation of animated sequences becomes a possibility. As the generation of the potentially substantial amounts of pen-and-ink drawings needed for animated sequences can really become tedious, we might even say computer support becomes a necessity. For this reason, we implemented visualization of fly-throughs ('fly-arounds') for 3D scenes that need to be rendered in pen-and-ink style.

(iv) Regarding distributed rendering techniques applied to supporting computer generation of images, various references can be found in the area where photo-realistic imaging is the issue. A good overview of the efforts related to this topic is given in [Green 91]. Supplementary to these efforts, it is of course equally possible to exploit distributed rendering techniques in the generation of (non-photo-realistic) pen-and-ink illustration. We realized this by distributing the work over a set of networked PC's.

In Section 2, the main techniques we implemented for realizing pen-and-ink illustration are elucidated. Section 3 describes briefly how we distribute the rendering over networked processors. Results are given in Section 4, whereas conclusions and directions for future research are given in Section 5.

2 Overview of the rendering algorithm

2.1 A 3D object representation

Although the drawing of pen strokes is inherently a 2D process, the utilization of underlying 3D object representations definitely has many advantages: (i) 3D modeling in combination with 2D projection automatically leads to correct perspective imaging; (ii) when viewing a model from various sides (of which animation is a special case), 3D information has to be present for efficiency, as otherwise a re-modeling needs to be done manually for each view, (iii) 3D models allow the inclusion of higher-level shading techniques, which can be of value for pen-and-ink illustration, even given the fact only bi-tonal drawing is available, etc. For these reasons, we opted to model the scenes to be rendered in a 3D object representation. One way to represent objects in 3D is by approximating them with planar polygons. Although this has it's limitations, a lot of scenes can be represented in an elegant and efficient way.

2.2 Hidden surface removal

Taking the 3D object model and an appropriate viewing transformation, it has to be decided which outlines and faces of the model are visible and which are hidden. In the literature, many kinds of hidden-surface removal/visible surface detection algorithms have been presented; good overviews of (and references to) these algorithms can be found in

[Sutherland74] and [Foley et. al 96]. The algorithm we opted for is a combined 3D/2D algorithm in the 'area subdivision' category. It is similar to the Weiler-Atherton algorithm [Weiler77]. The 'CompareSurface'-routine, as the core of our O (n^2) hidden surface algorithm, is explained in the following of this section.

'CompareSurface' is used to compare two arbitrary oriented 3D polygons A and B and to retain the projected parts -in the projected (2D) screen space- that are visible from the view point. The computational intensive clipping step involved in this process, which narrows down to a 2D Boolean difference between the projected polygons, is elucidated in the next section. In order to know which of the polygon(-part)s have to be subtracted and which not, the following steps are undertaken:

The intersection line of the planes in which the polygons A and B are defined is calculated. The projection of the intersection line defines where the mutual visibility of the two polygons changes: if polygon A is in front of polygon B on one side of the intersection line, it will be behind polygon B on the other side of the intersection line (and vice versa). A special case occurs when the two planes holding the polygons are parallel. Then the mutual order of the polygons is determined by the distance of their plane is to the view point. Note also that polygons perpendicular to the viewing direction are considered invisible.

In general, three non-degenerate cases can occur:
(i) When both polygons lie on different sides of the intersection line, the polygons do not overlap and no further processing is necessary.
(ii) One projected polygon, say A, is completely on one side of the intersection line, while the other, B, is split by the intersection line (or on the same side as A). In this case, it is sufficient to check the state of a single point in A in order to get the visibility information.
(iii) both projected polygons overlap the intersection line. In this case, both polygons are split. On the side of the intersection line where A is in front of B, the split part of A is subtracted from B (and vice versa).

2.3 Clipping: Boolean difference on 2D polygons

Before elucidating the principles used in the clipping-step of the rendering algorithm, we give some definitions :

- a loop is a linked list of points
- a polygon consists of an outer and zero or more inner loops
- the outer loop is oriented counterclockwise
- the inner loops (or holes) are oriented clockwise
- a polygon should not be self-intersecting

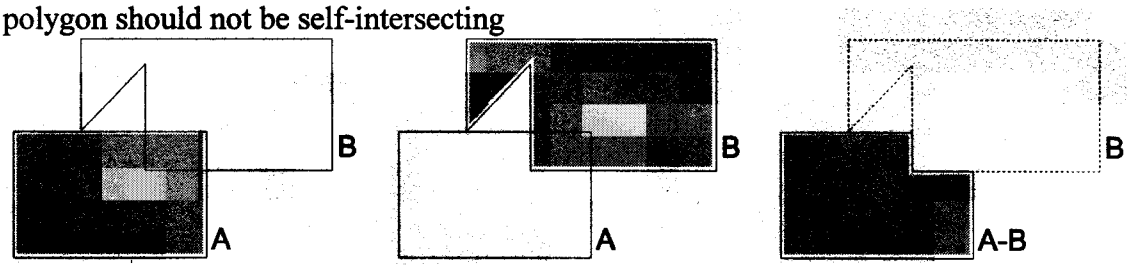

Figure 1. Boolean difference of two polygons

To subtract a polygon B from a polygon A (cf. Figure 1), resulting in one or more polygons C
(i) Find all intersection points between the 2 polygons and insert them in both polygons. For each intersection point, keep a pointer to the corresponding point in the other polygon. We implemented an O(n^2) algorithm for finding these intersections;

(ii) Mark all the points to be in one of 3 categories (cf. Figure 2):
- 'on' the other polygon (using an epsilon maximum distance as an approximation);
- 'inside' the other polygon;
- 'outside' the other polygon;

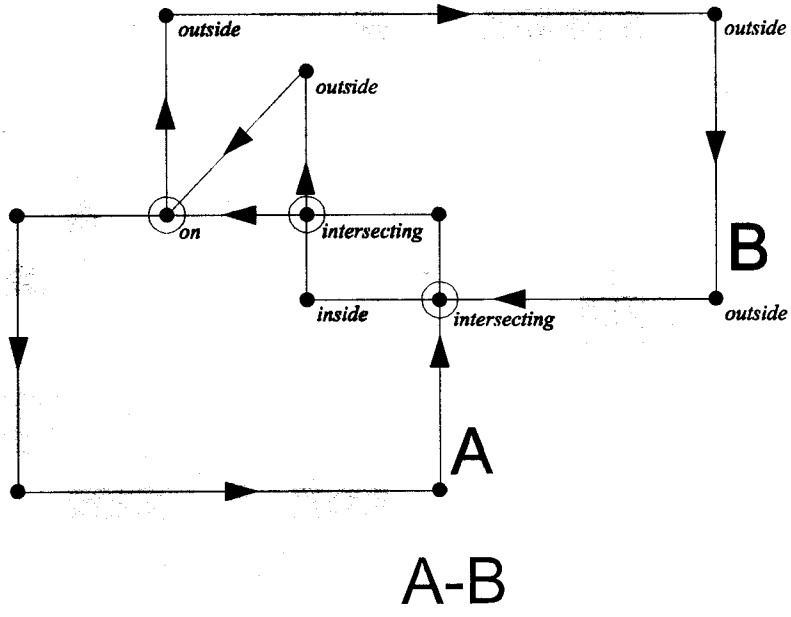

Figure 2. Marking of the polygon points

(iii) When two successive points in a polygon are on the other polygon, a point between them is also checked. If this new point is not on the other polygon, it is inserted into the polygon, dividing the edge at issue into 2 sub-edges.

(iv) Generate the loops of the result :
- Reverse polygon B (making the outer loop clockwise and the inner loop counterclockwise)
- A point of A is called a good point if it is outside of B. Points of B are good if they are inside A. The resulting polygon[s] will contain all good points, together with all the 'on' points.
- While there are good points :
  - Use such a point as a starting point.
  - While you don't reach an 'on'-point, nor your starting point, build a new loop by these points and mark the points as 'visited'. This loop will be part of the resulting polygon[s] C.
  - If you are reaching the starting point again, one of the resulting loop is finished.
  - If you reached an 'on'-point, switch over to the other polygon. If the first of the following points that is not 'on', is a 'good' point, stay in the new polygon. Otherwise use the last of the 'on' points to switch back to the first polygon.
- Special case to solve : all points of a loop are 'on' the other polygon and vice versa. This can only be if a loop of one polygon converges with a loop of the other polygon. If both loops have the same orientation, the loop will also be part of the resulting polygon (e.g. when polygon B is a hole of polygon A : subtracting it just preserves the hole in A). If both loops have the opposite orientation (one clockwise, the other counterclockwise), the loop will not be part of the result (e.g. when polygon A and polygon B are equal : the result is an empty polygon).

(v) Remove the auxiliary points that were introduced in step 3.

(vi) All counterclockwise loops of C are outer loops, all clockwise loops will be holes in one of the outer loops. A tree is build by checking one point of each loop to be inside/outside the other loops. This testing is optimized by using bounding box tests.

2.4 Pen texturing, -shading and rendering of strokes

As in conventional texture mapping, a pen-based texture is a pattern attached to a surface to be rendered. By choosing various patterns (of which bit-mapped images are an instance), one can influence various characteristics of the surface at issue: tone, structure, reflection, displacement, etc. When being restricted to purely pen-and-ink attributes, texturing becomes peculiar in comparison with the general texturing approach. The most obvious difference is of course the fact only two colors are present in the color pallet. In order to come to useful texturing and shading, appropriate half-toning alike concepts have to be utilized, but taking into account that the stroke primitives used in the pen-texturing normally influence relatively large amounts of pixels, rather than pixel (or even sub-pixel) primitives.

When drawing pen-textures in general, various aspects relating to the specific nature of pen-and-ink illustration have to be taken into account: the meaning of the elements in the picture, the utilization of outlines, the characteristics of the pen strokes used, the relation between texture and tone, resolution, orientation of the surfaces to be textured, etc. It would lead us too far if we need to tackle all these issues; the interested reader is referred to [Salisbury et. al. 94] and [Winkenbach et. al. 94].

The pen-textures we utilize are composed of multi-level strokes [Winkenbach et. al. 94], in which an increasing number of levels is used in order to create the desired tone for the surface at issue. In Figure 3, an example of this multi-level approach is depicted.

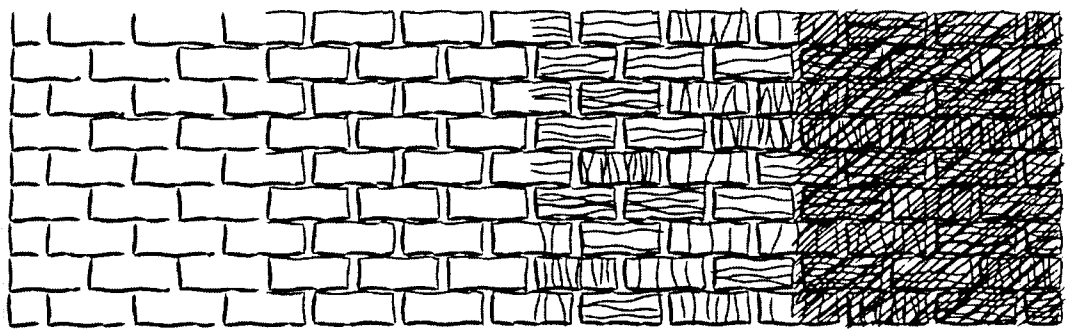

Figure 3. Multi-level pen-based texturing

The main difference between our technique and the former referenced one, is to be found in the statistical shading method we implemented :

Because the thickness of the strokes does not change when the polygon is tilted away from the view direction, each individual type of stroke will appear more black, depending on the viewing angle. We define the Blackness(i) of a stroke type i as a value varying from 0 for white to 1 for completely black. For example, when a brick wall is viewed from the side, the vertical strokes will almost completely dominate, while the horizontal strokes will have almost the same appearance as before. In this case, the vertical strokes will have a much higher blackness than the horizontal ones. If we want the wall to look just as black

independent of the viewing direction, we can randomly leave out some lines of the stroke. The more lines that are left out, the less black the stroke will appear. It seems a good idea to first leave out stroke types with a high blackness. This gives the picture a more equalized appearance. So, in our example it makes more sense to leave out the vertical strokes instead of the horizontal ones.

In a first step, all levels of strokes will be used, until the desired blackness is reached. The strokes of the last level that is tested, will not all be drawn. First the strokes are categorized depending on their type. The first of the four levels in Figure 3 consists of two types : longer horizontal and shorter vertical strokes to represent the outline of brics.

Suppose there are n different types of strokes in that current level. Let *TotalBlackness* be the sum of the blackness of all the types. And let *DesiredBlackness* be the degree of blackness that is needed for the current polygon. If *TotalBlackness* is smaller than or equal to *DesiredBlackness*, all the strokes will be constructed.

In the other case, only a certain number of the strokes will be drawn. To calculate this number, we introduce the term *Used (i)*. It represents the blackness for each type of stroke that is meant to be drawn. The chance factor *P(i)* controls which percentage of the strokes will be drawn. So $Used(i)=P(i)\cdot Blackness(i)$. This factor will be determined by the following algorithm.

The types of strokes are sorted on decreasing blackness. We want to reduce the number of strokes which have a greater blackness first. So the blackness of that type will be multiplied by a chance factor *P(1)* to become equal to the second largest share. In other words : $Used(1):=Blackness(2)$, thus $P(1):=Blackness(2)/Blackness(1)$. The total blackness that is generated with this new situation is again computed. If the desired blackness is still exceeded, the two largest shares are lowered so they are equal to the third largest share. This continues until the desired blackness is not reached anymore.

In this way, the first k types all will have a used blackness of *Blackness (k)*. Their chance of being accepted has been diminished, while for the other stroke types $i = k+1 \ldots n$, the chance is set to 1.

At the end, *TotalBlackness* does not reach *DesiredBlackness* anymore. The difference will be evenly distributed over the types that had a high *Blackness*, so they all will get the same new blackness.

The above described algorithm gives the opportunity to determine which stokes have to be drawn to get a certain degree of blackness before actually starting. It is a big benefit compared to having to calculate the blackness constantly while drawing.

For the actual generation of the strokes themselves, various techniques can be opted for; e.g. in [Hsu et. al. 94] it is illustrated how other attributes -texture, transparency, etc.- can be associated to curves acting as strokes, whereas in [Finkelstein et. al. 94] the representation, manipulation and rendering of curves for stroke rendering is elucidated. In our implementation, we use sinusoidal deformation of a n-sided flat polygon to represent strokes. These can either be generated into a PostScript format or can be directly rendered into a bitmap.

As a final point in this section, we would like to mention that we also adapted the clipping of the strokes to be rendered. Indeed, a fully 3D rendering method would most probably treat strokes in a fashion depicted in Figure 4A, whereas our approach enables clipping as shown

in Figure 4B. The endpoints of the strokes are randomly translated to get rid of the effect that all strokes end in a straight line.

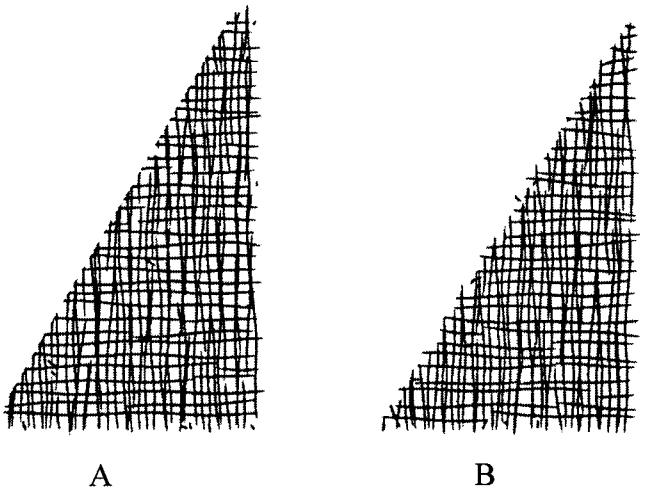

Figure 4. Appropriate clipping of strokes: bad way (A) and good way (B)

3 Distribution of the rendering

In order to speed up the rendering process, we decided to utilize the processing power available in a network of Pentium PC's. This idea clearly is not new, but its realization certainly helps in getting the images rendered faster.

In order to realize the distributed rendering architecture, we implemented a client-server based approach, in which a central master distributes work to slaves available in the network. The number of slaves in the network can vary, and can be changed dynamically.

When distributing the work for the pen-and-ink pipeline, various options can be taken:
- distribution of the work on an image by image basis: this type of distribution gives a linear performance gain, but is only useful in cases where multiple viewpoints of the scene have to be generated (e.g. for animated sequences);
- distribution scheme in which various processors work on one image: this is the most promising approach, since one can profit of the multiple processors available in the generation of a single image as well. Two main parts of the pen-and-ink pipeline are prone to being distributed: the calculations in the clipping/Boolean difference step and the rendering of the strokes. In the current version of the algorithms, we distributed the calculations regarding the rendering of the strokes.

In this approach, the clipping/Boolean difference step is performed by the master in a preprocessing step. After this preprocessing step, the master maintains a pool of polygons to be generated with strokes and distributes dynamically the definition of the polygons to be generated to the processors in the network. Each available processor in the network (that has granted being used for the calculations) receives the definition of requested polygons and renders them into a bitmap. When all polygons have been rendered by the slaves, the slaves send the resulting bitmap to the master where the bitmaps at issue are merged (OR-ed or AND-ed depending on the values representing black and white). This approach leads to a quite acceptable load balance, but still can be improved. As all slaves end their work at different moments in time, the master can combine the resulting bitmaps while he gets them.

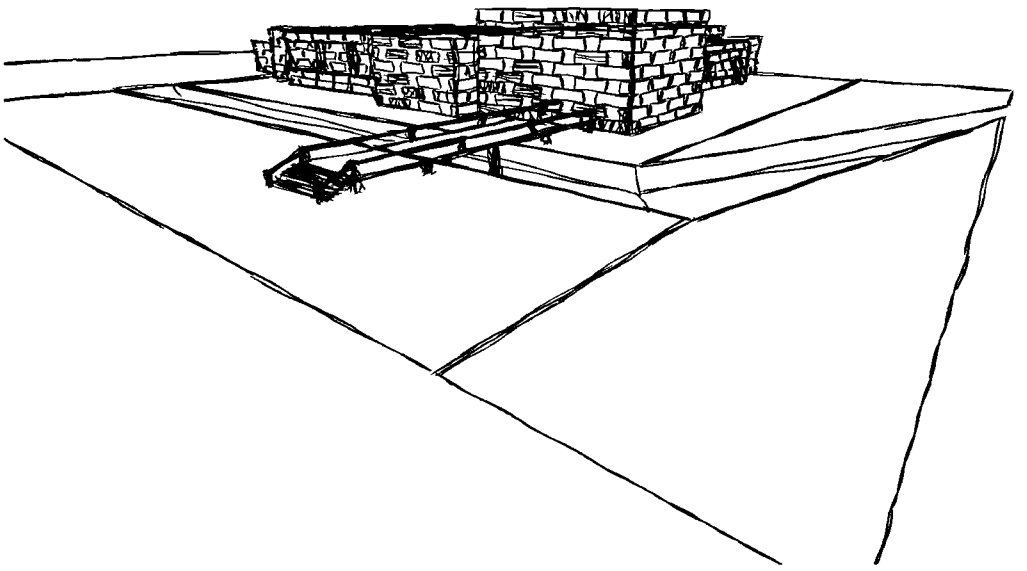

Figure 5. 'Stylized' version of our building in pen-drawn outlines and texturing

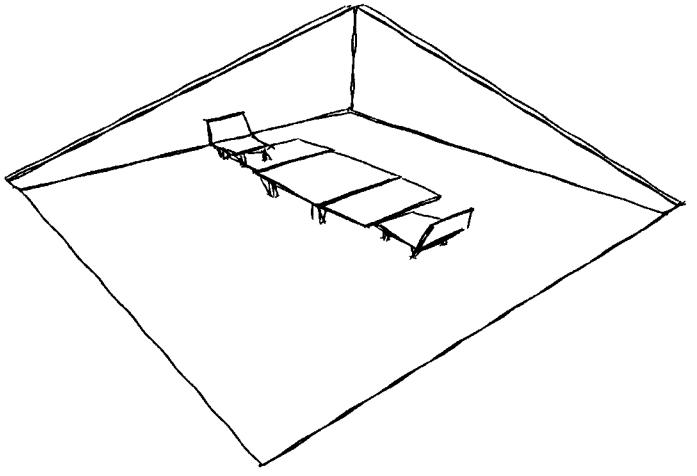

Figure 6. Outline only version of the scene utilized in the test-run

4 Results

Figure 5 shows a snapshot in which a stylized version of our building has been modeled.

In figure 6, the scene is depicted that we used in our test-run on the distributed implementation. In the test-run, a resolution of 768 x 576 pixels was used. This scene looks rather simple, but in first instance we wanted to test the parallel rendering of complex strokes. In general there is much more time spend on the fitting and clipping of the strokes than on the hidden surface removal.

The preprocessing of the clipping/Boolean difference operations for this scene took 20 seconds on the master processor. We tested the scene with various stroke types, the average generation times for the strokes are the following:

Sequential version : 50 sec.

Distributed version:

1 slave:	58 sec.
2 slaves:	29 sec.
3 slaves:	18 sec.
4 slaves:	15 sec.

This times include the distribution of the polygons to be stroked, the stroking on the slaves and finally the combining of the bitmaps, done by the master.

One can see a slight overhead is introduced by distributing the algorithm. This overhead is fully due to the needed communication over a 10 Mbit/s Ethernet network. Once this overhead is accepted, our test run shows that an almost linear speed-up is gained when using up to four slaves in the distributed rendering. In the near future, we hope to be able to test run on more processors.

5 Conclusions and Future Research

In this paper, we presented the techniques we designed and implemented in order to generate computer-based drawing of images looking alike traditional pen-and-ink illustrations. The main algorithms of the pen-and-ink rendering pipeline have been elucidated and we explained how a networked version was realized. Results have been given and discussed.

Several directions of future research can be indicated:
- Inclusion of curved surfaces in the object representation is promising area in which various 'hard-to-tackle' problems with respect to pen-and-ink illustration are present. A simple example of this is a round tower. When we approximate the tower by polygons, the curvature should also be applied to the texture;
- Realization of more advanced illumination models (effective implementation of shadowing) and texture indication (i.e. incorporation of interactive techniques reported upon in [Salisbury et. al. 94]) are possible extensions towards the future as well;
- Research and Development around new styles of pen-based texturing, in which more advanced procedurally defined texturing techniques could be very promising;
- Optimization efforts: since the Boolean difference operation is a time-expensive operation, optimization efforts should be directed towards this part of the overall algorithm:
(i) sorting the polygons front to back in the clipping stage will more quickly detect cases in which polygons are fully covered by other polygons (and hence more quickly terminate the Boolean difference operation)
inclusion of a Z-buffer hidden-surface removal technique in order detect more quickly polygons that are fully covered by other polygons
(ii) on a pragmatic level, porting to SGI hardware in order to exploit the SGI rendering hardware in some steps of our rendering algorithms can be a target for future work (since all algorithms have been implemented in ANSI C, this job shouldn't be too hard).

Acknowledgments

Part of this work is funded by the Flemish Government, by the European Regional Development Fund and by the National Fund for Scientific Research (NFWO) in Belgium. We would like to acknowledge the people at the Expertise centre for Digital Media who contributed in one way or another to this work; especially the help of Koen Elens and the implementation assistance of Kurt Dethier is greatly appreciated. Paul Akkermans is acknowledged for the construction of the 3D models.

References

Cockshott T., J. Patterson and D. England, "Modelling the Texture of Paint", in *Proc. Eurographics'92*, pp. 217-226, 1992.

Finkelstein A. and D.H. Salesin, "Multiresolution Curves", in *Computer Graphics* 28 (4), pp. 261-268, 1994.

Foley J.D., A. Van Dam, S.K. Feiner and J. F. Hughes, *Computer Graphics, Princliples and Practise*, 2nd Edition, pp. 649-717.

Green S., *Parallel Processing for Computer Graphics*, Pitman, London, 1991.

Haeberli P., "Paint by Numbers: Abstract Image Representations", in *Computer Graphics* 24 (4), pp. 207-214, 1990.

Hsu S.C. and I.H.Lee, "Drawing and Animation Using Skeletal Strokes", in *Computer Graphics* 28 (4), pp. 109-118, 1994.

Klitment S., *Architectural Sketching and Rendering: Techniques for Designers and Artists*, Whitney Library of Design, New York ,1984.

Litwinowicz P.C., "Inkwell: A 2 ½D Animation System", in *Computer Graphics* 25 (4), pp. 113-121, 1991.

Salisbury M.P., S.E. Anderson, R. Barzel and D.H. Salesin, "Interactive Pen-and-Ink Illustration", in *Computer Graphics* 28 (4), pp. 101-108, 1994.

Strassman S., "Hairy Brushes", in *Computer Graphics* 20 (4), pp. 225-232, 1986.

Sutherland I.E., R.F. Sproull and R.A. Schumacker, "A Characterisation of Ten Hidden-Surface Algorithms", *ACM Computing Surveys* 6 (1), pp. 1-55, 1974.

Van Reeth F., "Integrating 2 ½ D Computer Animation Techniques for Supporting Traditional Animation", in Proc. Computer Animation'96, pp. 118- 125, 1996.

Winkenbach G. and D.H. Salesin, "Computer-Generated Pen-and-Ink Illustration", in *Computer Graphics* 28 (4), pp. 91-100, 1994.

Weiler K. and P. Atherthon, "Hidden Surface Removal using polygon area sorting", in *Computer Graphics* 11 (2), pp. 214-222, 1977.