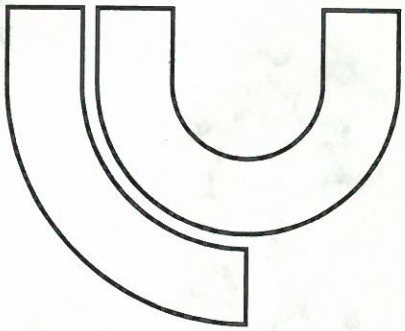
A study of global illumination models and their use in a transputer based parallel system

Supplementary material



Limburgs Universitair Centrum

Faculteit Wetenschappen

AN EXTENSIBLE TWO-DIMENSIONAL ANIMATION/PAINT SYSTEM

Bijstelling bij het proefschrift voorgelegd tot het behalen van de graad van

Doctor in de Wetenschappen, richting Informatica

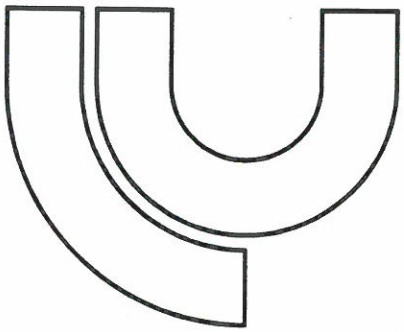
aan het Limburgs Universitair Centrum te verdedigen door

WIM LAMOTTE

Promotor : Prof. Dr. E. Flerackers

Copromotor : Prof. Dr. T. D'Hondt

1994



Limburgs Universitair Centrum

Faculteit Wetenschappen

AN EXTENSIBLE TWO-DIMENSIONAL ANIMATION/PAINT SYSTEM

Bijstelling bij het proefschrift voorgelegd tot het behalen van de graad van

Doctor in de Wetenschappen, richting Informatica

aan het Limburgs Universitair Centrum te verdedigen door

WIM LAMOTTE

Promotor : Prof. Dr. E. Flerackers

Copromotor : Prof. Dr. T. D'Hondt

1994

An Extensible Two-dimensional Animation/Paint System

Abstract

In this text, we present an Extensible Two-dimensional Animation/Paint System, called X-Stream. X-Stream is designed as a testbed for integrating various animation and painting techniques. Utilizing a uniform internal representation (and accompanying operations on it) for all geometric primitives, X-Stream can integrate direct manipulation-, key-framing based-, as well as procedurally defined animation techniques.

A key item of the system is that it can be expanded in a modular way using the standard programming environment feature of dynamic linking. By making the uniform internal representation accessible externally (by means of an abstract data type), a close interaction between the basic system and the added functionality is guaranteed.

1. Introduction

Classifying computer animation systems, one can distinguish between computer-assisted animation and modelled computer animation [Magnenat-Thalmann 85]. In the former class, the computer is merely used as a tool to perform simple operations (e.g. filling an area), while the animator/operator does most of the animation work 'by hand'. In the latter class, the computer is used as a fully integrated tool, performing and supporting most of the animation and rendering work. The animator can work on a higher level by controlling and shaping appropriate animation parameters. Computer-assisted animation systems are often found in the 2D animation area, whereas modelled animation systems are mostly encountered in the 3D animation domain. With X-Stream, we believe to having developed a hybrid computer-assisted / modelled animation and paint system. The main underlying design criteria and objectives are elucidated below:

1) With the development of X-Stream, we tried to get an idea of how the techniques one normally tends to encounter in a three-dimensional animation system can be applied in two-dimensional animation/paint systems (hence trying to get the best of both worlds). This means that beside interactive and direct drawing manipulations [Litwinowicz 91], the notion of modelled and animated geometry is provided. E.g., an airbrushing operation can interactively be drawn on a single background bitmap, but it can just as well be drawn on a sequence of images, following some primitive geometric entity (a line, a polygon, an open or closed spline curve, ...) which in its turn can potentially be defined within an hierarchy being animated in the sequence at issue. Key-framing based methods, as well as scripted techniques [Reynolds 82] have to be provided.

2) The user interface principles should be known and intuitive to a wide range of potential users. In order to realise this, we took recourse to a Windows-based user-interface, as it is becoming the de facto standard in PC-environments (and the PC is our targeted platform, as it hosts our real-time display board; cf. Section 6).

3) We wanted X-Stream to be an extensible system. It is often the case that for a given animation production, the required functionality one needs, or the specific tool which could significantly speed up the realisation of a certain animation sequence, is missing. An open-ended animation system should enable that missing functions can be incorporated. Moreover, it should be possible that new techniques reported upon in the literature -see also some of the examples in the paper- can be incorporated into the system by a programmer in a modular way. The realization of state-of-the-art animations often requires the combination of animators' skills as well as those of programmers, so the system should support this.

4) X-Stream should serve as a testbed for developments in 2D animation and painting techniques (in fact, project students greatly appreciate the fact that their escapades in 2D -or $2^{1/2}$D- graphics and animation can be incorporated in a modular way within an overall structure; moreover, they don't have to re-invent the wheel in terms of menu/user interfacing structures, import/export of data, the basic manipulation of bitmaps and geometry, etc.).

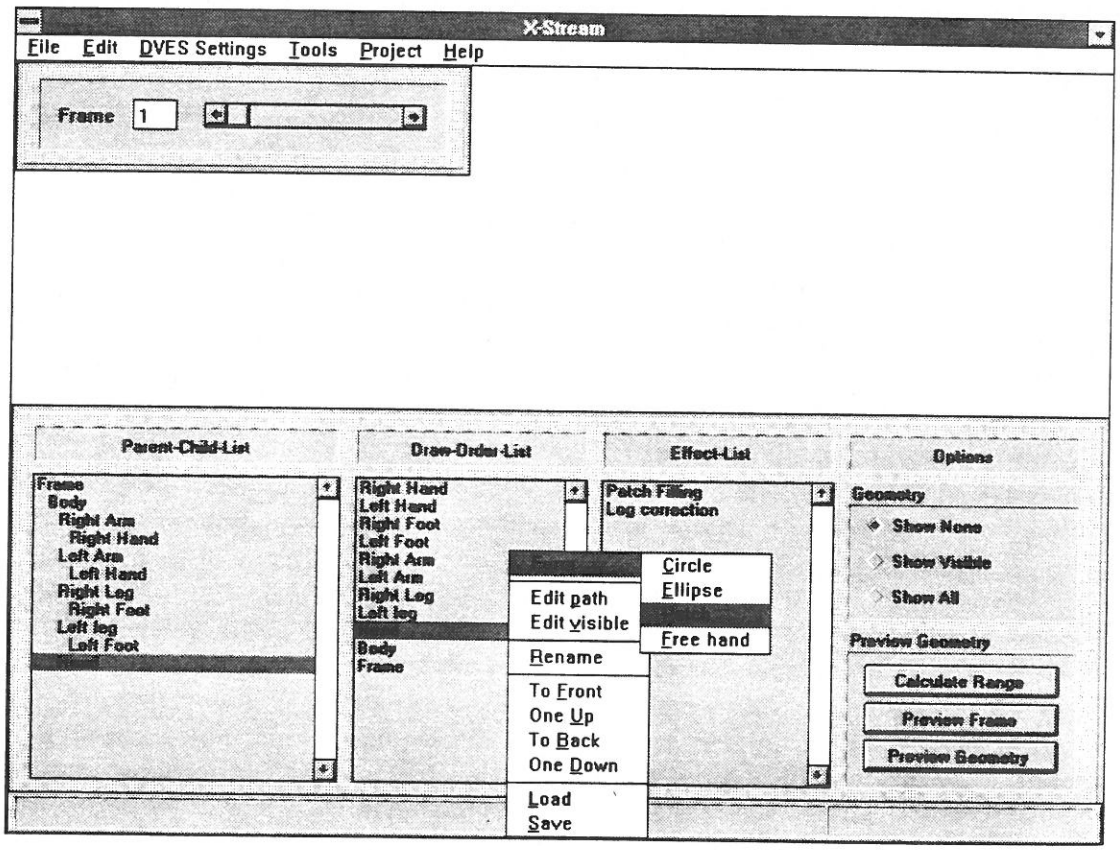


Figure 1 : X-Stream's main user interface

Fig. 1 depicts the main user interface screen of X-Stream. As the manipulation of data over time in general is the principal goal of the system, a tool to easily select parameters on a given time in the animated sequence is provided by means of the top-left scroll-bar. The bottom window is the main control window. In here, the hierarchy of geometries can be created, forms can be associated with geometry, the drawing order can be defined, effects can be imposed and a preview of the animation can be created. The modelled animation items can be

created, selected and manipulated interactively on a second (video-)screen as well as in a pop-up drawing window. The menu bar at the top houses the basic system functions as well as user-extensible functionalities. Context-sensitive functions (drawing primitives, ordering,...) are invoked through pop-up menu's, as shown in the figure.

Section 2 highlights the basic ideas concerning drawing with geometry, whereas Section 3 discusses the way in which bitmaps are manipulated. Section 4 explains how the extensibility of X-Stream is realized using standard functionalities of today's programming environments; some examples are also given. Section 5 elucidates how animators/operators can use function curves as a way to influence animation/paint parameters. A few notes on the real-time display hardware are given in Section 6. Finally, conclusions and directions of future research can be found in Section 7.

2. Drawing with geometry

In order to have a system which can be used in a direct manipulation fashion as well as in a (scripted) modelled fashion, a uniform internal structure for geometric primitives is created. For this purpose, all geometric primitives - lines, polylines, rectangles, polygons, circles, ellipses, open or closed free-form curves, text, ... - are internally represented uniformly as splines. We are aware that this has some consequences at the performance level of some of the underlying algorithms, but the benefits of a performance gain certainly don't surpass these of the uniformity of the representation. The primitives can be changed by moving the appropriate control points (either interactively or by some piece of code).

Just like most of its 3D animation counterparts, X-Stream is equipped with hierarchical object specification, as well as with an infrastructure of transformation channels (translate-rotate-scale-skew-...) for motion specification (i.e. geometry serves as a path along which animated primitives can move). In two-dimensional animation, however, these linear transformation techniques are surpassed often by interactive free-form deformation techniques, accompanied by interpolation methods at in-between positions. In order to keep the interpolation algorithms straightforward, the various instances of the geometries to be interpolated at the respective key-frames are internally linked to each other by means of a one-to-one mapping of their control points. As a consequence, an insertion or deletion of a geometric primitive (or one of its control points) at a certain key-frame is chained through over all the key-frames at which an instance of the primitive at issue is present.

Geometries can have various standard attributes : they can be open or closed; the spline can have different orders of continuity at the control points; the outline can be drawn in a specific style and colour; they can represent a function (cf. Section 5); a closed geometry can be filled (with a uniform colour, an interpolated colour, a bitmap, a moving bitmap); they can be turned on and off with respect to drawing (rendering); etc.

The various geometric primitives have a drawing order, specified by their order in the second scroll-list "Draw-Order-List" (cf. Fig. 1). Geometries to be generated by code (scripts) are invoked by putting their name and associated parameters in the geometry list at the required position.

The internal operations on geometry are hidden behind an Abstract Data Type (ADT). We enlist some of the primitive operations to give an impression of how this is realized (this ADT also plays a major role in the extensibility of X-Stream; cf. Section 4):

```
HANDLE SPLINE_InitCurve (void);
void SPLINE_FreeCurve (HANDLE Curve);
...
BOOL SPLINE_CurveClosed (HANDLE Curve);
void SPLINE_CloseCurve (HANDLE Curve);
void SPLINE_SetFillStyle (HANDLE Curve, int FillStyle);
...
void SPLINE_InsertControlPoint (HANDLE Curve, int PrevPoint, int x, int y);
void SPLINE_DeleteControlPoint (HANDLE Curve, int Point);
void SPLINE_GetControlPoint  (HANDLE Curve, int Point, int *x, int *y);
void SPLINE_MoveControlPoint  (HANDLE Curve, int Point, int dx, int dy);
void SPLINE_SetControlPointType (HANDLE   Curve, int Point,
  int CuspSmoothSymm);
int SPLINE_GiveNumControlPoints (HANDLE   Curve);
...
HANDLE SPLINE_CopyCurve (HANDLE Curve);
...
HANDLE SPLINE_CreateRectangle (int x1, int y1, int x2, int y2);
HANDLE SPLINE_CreateCircle    (int x, int y, int r);
...
HANDLE SPLINE_Interpolate (HANDLE Curve1, HANDLE Curve2, double
  Percentage);
...
void SPLINE_DrawCurve (HW_DC *Dc,   HANDLE Curve);
void SPLINE_DrawSegment (HW_DC *Dc, HANDLE Curve, int LeftRight, int
  Point);
void SPLINE_ScanConvert (HW_DC *Dc, HANDLE Curve);
...
```

Colour Plate 1 shows some examples of geometric primitives.

In addition to the fact that geometries enable us to incorporate modelled animation techniques, they enable us to "render" the image at any resolution (when rendering at higher resolution, the used bitmaps -e.g. for backgrounds - should of course also be scanned or grabbed at a higher resolution). Pre-filtering anti-aliasing techniques can moreover be used for drawing the geometric primitives at high quality.

We finalize this section by stating that we are currently investigating how to incorporate in X-Stream a 2D version of our implementation of the 3D animated free-form geometry deformation technique [Sederberg 86], [Coquillart 91].

3. Manipulating bitmaps

Of the basic operations one encounters in "bitmap-manipulating" applications which are provided in X-Stream (aside the aforementioned geometry-related drawing), the brushing operation is probably the most important one. As in the geometry case, brushes can also be accessed through an ADT. The functions in this ADT are in their turn based on a more basic ADT (which, again, plays an important role in the extensibility of operations):

```
void FDRAW_PutPixel (HW_DC *Dc, int x, int y, COLORREF color );
void FDRAW_GetPixel (HW_DC *Dc, int x, int y, COLORREF *color );
...
void FDRAW_DrawLine (HW_DC *Dc, int x1, int y1, int x2, int y2 );
...
void FDRAW_SetDrawMode (HW_DC *Dc, int DrawMode, double Alpha );
...
void FDRAW_SetDC (HW_DC *Dc );
...

void FDRAW_GetPic (HW_DC *Dc, TBitmap *buffer, int x1, int y1, int x2,
  int y2 );

void FDRAW_PutPic (HW_DC *Dc, TBitmap *buffer, int x1, int y1, int x2,
  int y2 );
```

HW_DC is a struct which holds the current device context (current position in frame store, active bitplane mask, active display buffer, ...).

By integrating the geometry ADT and the bitmap ADT, interesting combinations result. E.g., the parameters governing the image manipulation (taken from [Holzmann 88]) in Colour Plates 3 to 5 that were applied to the original image in Colour Plate 2, can be animated using geometrical primitives representing spatial and scalar values.

4. Extensibility using DLL's and ADT's

General

Extending the functionality of applications in general often narrows down to developing additional data structures and writing primitive functions operating on these data structures. Using traditional compiler/linker methodology, this implies the newly developed functions (and related data structure definitions) have to be compiled in an object code format, and consequently have to be linked with the application code at issue. The process of statically relinking the application in fact creates a new executable application code. This clearly prohibits an easy extendibility of applications. Another alternative to extending the functionality of applications would be to develop separately executable modules which can 'loosely' be called (executed is a better term) from within the original application; clearly, this is even worse as a close sharing (and passing) of data between the separate execution modules and the 'main' application is not possible.

To circumvent this problem, the concept of dynamic linking is available in some of today's programming environments. Rather than linking the functions statically beforehand, the functions are loaded dynamically by the environment at run-time. Thus applications can be extended and updated without having to change the application's executable code. Microsoft Windows is such an environment supporting the concept of Dynamic Link Libraries (DLL's). As X-Stream is implemented under Windows, we took recourse to its Dynamic Link Library feature in order to have an extensible system.

Concretely, the expandability is realized following two issues:

- the ADT's used by us to develop the basic system are also put at the disposal of developers realizing supplementary DLL's;

- the menu holding "tools", "filters" and "effects" can be extended with new entries (invoking the new functionalities) following a simple protocol of DLL-binding.

We elucidate the binding using the bitmap-filter case : given the following declarations:

```
typedef void(FAR *TFilterFunc) (TBitmap *bitmap);
char DLLName [MAX_LEN_DLL_NAME];
char DLLProcName[MAX_LEN_PROC_NAME];
HINSTANCE hlib;
TFilterFunc lpFilterFunc;
```

the following piece of code calls a function in a DLL:

```
hlib = LoadLibrary (DLLName);
if ( hlib > HINSTANCE_ERROR )    // DLL loaded successfully
{
   lpFilterFunc = (TFilterFunc) GetProcAddress( hlib, DLLProcName );
   if ( lpFilterFunc != NULL )
       (*lpFilterFunc) (bitmap); // call to DLL function
   FreeLibrary (hlib);
}
```

This technique can be used to expand the system functionality. Indeed, when the name of all DLL's to be loaded are passed to X-Stream (by means of an ASCII file), a pull-down menu hosting a corresponding entry for each DLL is created. The names of callable functions within the DLL can be passed in a similar way (thus, by means of an ASCII file), they can be used to build a list of "lpFilterFunc"s which are called when the corresponding menu entry is selected.

User interfacing driven operations

As DLL's can become master over the message event loop, the user-extended functions are not limited to "batch"-processes:

(i) functions needing alternative manipulation methods for geometry,

(ii) extensions needing the activation of pop-up menu's for the initialisation of relevant parameters, or

(iii) operations in general needing the mouse position as input, can just as well be developed.

Example 1: Pseudo-physics

We mentioned previously that the geometrical primitives can be used as paths to be followed by other primitives.

Depending on the animation task at hand, the interactive drawing of these "motion geometries" can be a tiresome process. In order to circumvent this problem, one can take recourse to scripting techniques. By way of an example we implemented a DLL for realizing motion paths by means of pseudo-physics of flows (sink flows, source flows, vortex flows, uniform flows and their combinations) according to [Wejchert 91]. Figure 2 depicts some generated paths.

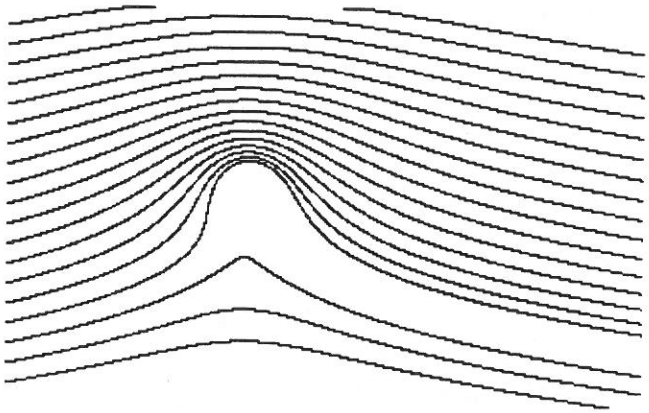


Figure 2: Using DLL's for specifying scripted motion paths

Example 2: Constraint-based animation

We mentioned previously that geometrical primitives in X-Stream can be arranged into a hierarchy in order to be animated accordingly. As a counterpart to this way of animating, we studied the work of Van Overveld in the area of force-driven, non-hierarchical constraint-based animation [Van Overveld 90] and implemented a DLL realizing his physics-based procedural animation technique.

Example 3: Image morphing as an hybrid bitmap/geometry operation

Warping and Morphing of bitmapped images are nowadays 'successful' effects. Originally not provided in X-Stream, they can be realized as a hybrid geometrical/bitmapped function. Indeed, using the aforementioned bitmap and geometry related ADT's, one can easily implement these effects [Wolberg 90] in a DLL. Colour Plate 1 shows an example of a interactively manipulated geometrical grid being used as a control grid for the bitmap deformation.

An alternative approach to make the system extensible would be to attach an interpreted language instead of using DLL's. However, our experience with CAEL [VanReeth 90], and with interpreted languages in general, makes us leave this option, since we loose an order of magnitude in performance.

5. Curves as functions

Indeed, by constraining the shape of a geometric curve, one can make sure that given a certain value within a domain, one gets the corresponding function value. A special type of curve editor is created for this purpose; Figure 3 shows an example of such a curve function. This curve function feature (taken from our visual programming environment CAEL [Van Reeth 90]) enables animators to control animated parameters without having to take recourse to a conventional programming language.

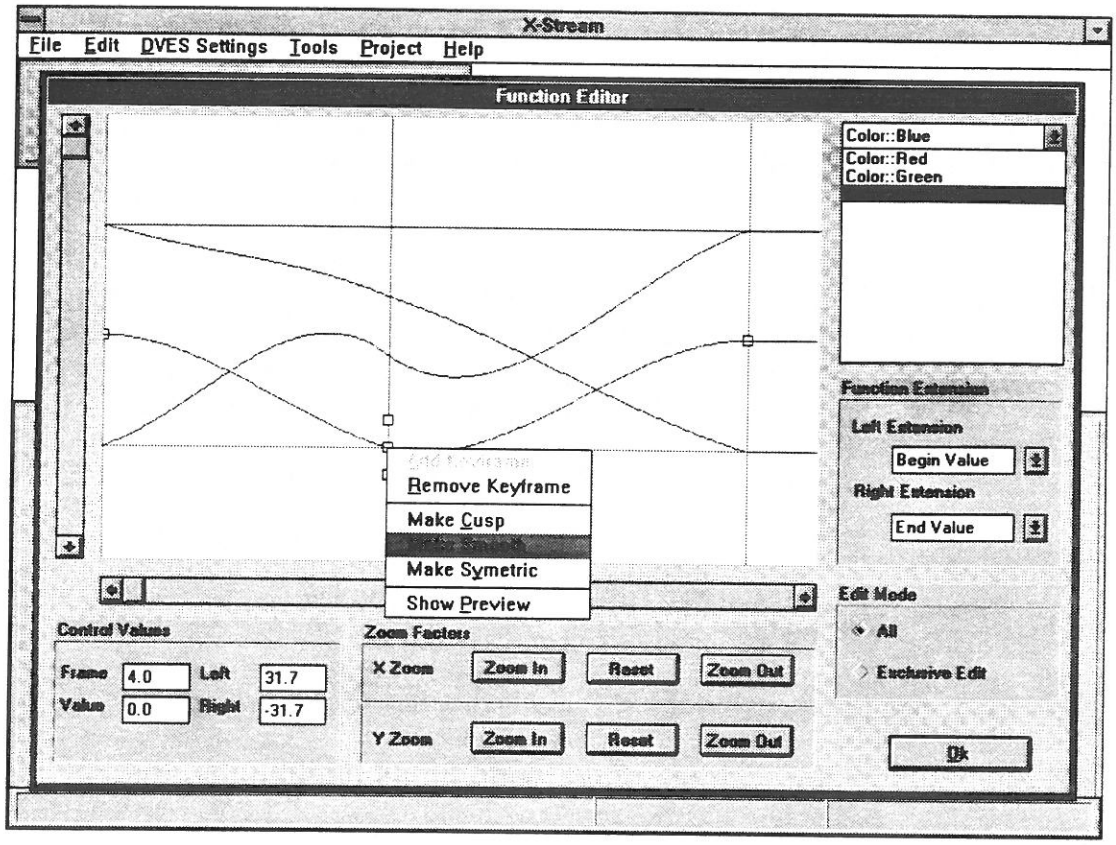


Figure 3: Curves can act as functions for controlling animated parameters

Curve functions can be called from within user-written DLL's by means of ADT calls (in which case they should be looked upon as being interpreted functions).

6. Hardware

X-Stream is running on a PC under MS-Windows. Aside the normal SVGA graphics card (on which the user interface is displayed), an advanced graphics video board is utilized for displaying the animated sequences in real-time. The board hosts DRAM storage from which full-resolution (768 X 576) true-colour images can be displayed in real-time (for the video standard used in Europe : 25 images per second). Using 4 MB SIMMs, we have 64 MB RAM at our disposal; using 16 MB SIMMs, we have 256 MB RAM at our disposal, implying about 2 seconds resp. 8 seconds of animation can be displayed without having to go to external storage. Depending on the envisioned application domain, it might be possible that this amount of memory is not sufficient, so we are currently investigating how current off-the-shelf disk technology (in the form of RAID's - Redundant Arrays of Inexpensive Disks) can be used to deliver a satisfactory throughput in order to achieve real-time display.

7. Conclusions and future research

We have described the key issues of X-Stream, our testbed for developments in 2D animation and painting. It is discussed how we realize the concept of an open-ended system utilizing standard programming features.

Looking at the various sections in this paper, one can state X-Stream is a hybrid system along at least three dimensions: it has direct drawing features on the one hand, and modelled geometries on the other hand; it has keyframed animation techniques on the one hand, and scripted procedural techniques on the other hand; it can be extended using compiled DLL's on the one hand, and interpreted function curves on the other hand.

Our future directions of research around X-Stream involve:

- realization of application-domain specific DLLs
- Mixing 2D and 3D animation by incorporating 3D animation data (image data, depth buffers, normal values, etc.) from our 3D animation station [Van Reeth 91], [Lamotte 93].
- On the painting level, effects and techniques alike the ones reported upon in [Haeberli 90] and [Strassman 86] are likely to be incorporated as potential extension on X-Stream.

Acknowledgements

We would like to thank all the people involved in the realization of the X-Stream project. Especially the assistance in the implementation of Kurt Dethier, Jochen Grosemans and Philip Schaeken is greatly appreciated. Luc Jorissen is acknowledged for his assistance on the hardware level.

References

Coquillart, S. and Jancène, P. "Animated Free-Form Deformation: an Interactive Animation Technique", **Computer Graphics**, Vol. 25, No. 4, 1991, pp. 23-26.

Haeberli, P., "Paint By Numbers : Abstract Image Representations", **Computer Graphics**, Vol. 24, No. 4, 1990, pp. 207-214.

Litwinowicz, P.C., "Inkwell: A $2^{1/2}$-D Animation System", **Computer Graphics,** Vol. 25, No. 4, 1991, pp. 113-122.

Holzmann, G. "Beyond Photography", Prentice Hall, 1988.

Lamotte, W., Van Reeth, F. and Flerackers, E., "Computer Graphics and Animation on a Transputer Platform". In: Grebe, R. et al., "Tranputer Applications and Systems '93 - Vol. 1" (Proceedings World Transputer Congress '93), IOS Press, Amsterdam Oxford Wahington Tokyo, pp. 233-243.

Magnenat-Thalmann, N. and Thalmann, D. "Computer Animation, Theory and Practice", Springer-Verlag, Tokyo Berlin Heidelberg New York, 1985.

Reynolds, C., "Computer Animation with Scripts and Actors", **Computer Graphics,** Vol. 16, No. 3, 1982, pp. 157-166.

Sederberg, T. and Parry, S., "Free-Form Deformation of Solid Geometric Models", **Computer Graphics,** Vol. 20, No. 4, 1986, pp. 151-160.

Strassman, S., "Hairy Brushes", **Computer Graphics,** Vol. 20, No. 4, 1986, pp. 225-232.

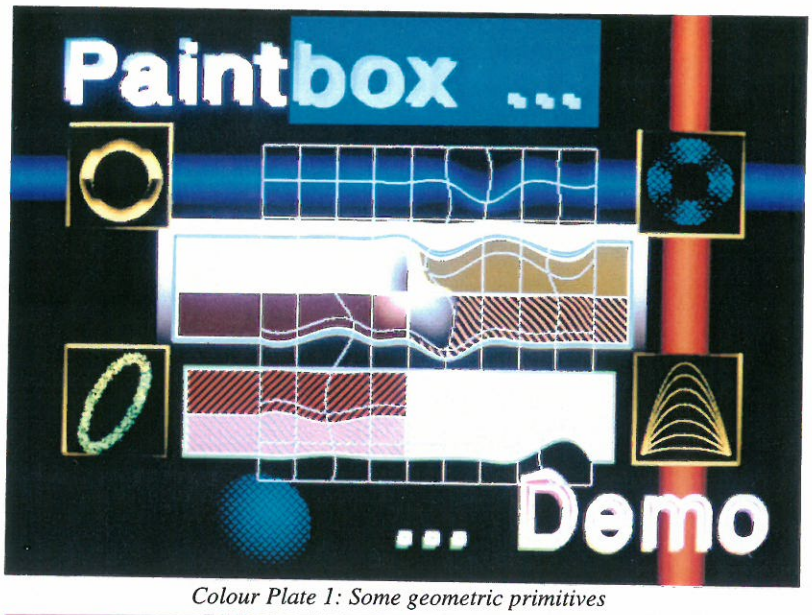
Van Overveld, C.W.A.M., "A Technique for Motion Specification in Computer Animation", **The Visual Computer**, Vol. 6, No. 2, pp. 106-116, 1990.

Van Reeth, F. and Flerackers, E., "Visual Programming in a Computer Animation Environment", In: Proc 6th IEEE CS Workshop on Visual Languages, pp. 194-199, 1990.

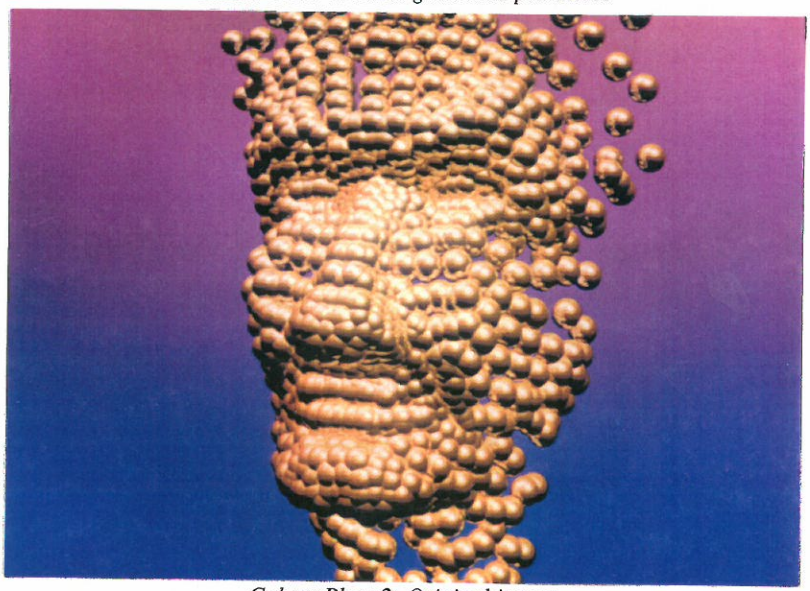
Van Reeth, F. and Flerackers, E., "Utilizing Parallel Processing in Computer Animation", In: Proc CA'91, pp. 227-240.

Wejchert, J. and Haumann, D. "Animation Aerodynamics", **Computer Graphics**, Vol. 25, No. 4, 1991, pp. 19-22.

Wolberg, G. "Digital Image Warping", IEEE Computer Society Press, Los Alamitos, 1990.

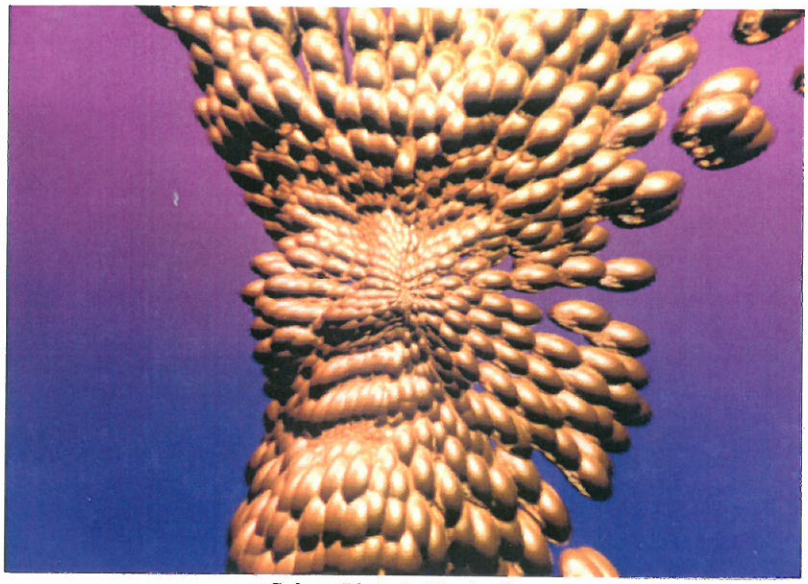


Colour Plate 1: Some geometric primitives

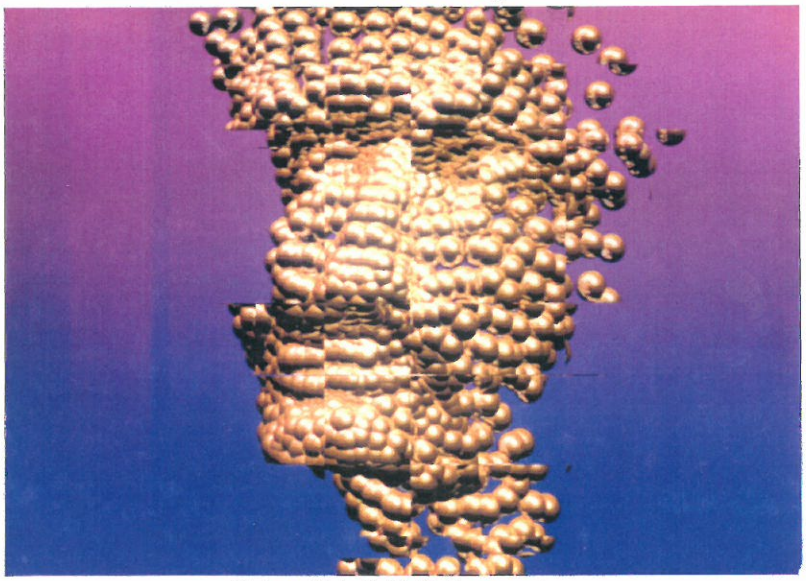


Colour Plate 2: Original image

Colour Plate 3: Oil painting effect



Colour Plate 4: Pinch effect



Colour Plate 5: Shifted rectangles