Vision, Video, and Graphics (2003)
P. Hall, P. Willis (Editors)

Quasi-3D Cel-based Animation

M. Qi and P. J. Willis

Department of Computer Science, University of Bath, Bath, UK

Abstract

We present a method for image compositing and rendering using 2D geometric shapes or raster images as input primitives, disposed in a 3D environment around which the camera can move. An animation system has been implemented which calculates camera and scene information to render the frames. The key features of this quasi-3D system are described. Two animations generated by the system are given as examples.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Quasi-3D, Composing and Rendering, Computer Animation

1. Introduction

Computer animation systems are typically 3D, mimicking modelled animation such as claymation. These can offer sophisticated lighting and rendering options but are also compute-intensive. There are also some 2D systems, which emulate hand-drawn cel animation. These naturally offer a way of building the output image, layer by layer, but do not take full advantage of image composition.

Lengyel and Snyder[1]presents an image composition system based on Microsoft's Talisman hardware architecture[2]. As their main concern is speed rather than image quality, image resolution is limited. Tic-Tac-Toon[3] focuses on every stage of the traditional animation process. With vector primitives, no digitized data can be used as input. Some commercial systems, such as Cambridge Animation Systems Animo product, combine vector line drawing with 3D models.

We have explored a different approach to this 2D/3D area. We have built a system which uses 2D pictures, located on cels (2D planes) which can be disposed anywhere in 3D space. In other words, we have replaced the traditional stack of cels found in a 2D system with one in which cels become modelling elements in a 3D world. The camera can move around this world, giving the same freedom of view as a full 3D system. This permits us to build 3D-like worlds, without the need for full 3D modelling. To be effective, the system must treat the cels as input images, which are to be manipulated and then rendered to an output picture. The technology to do this thus sits between pure synthetic graphics (where pictures are only output elements) and image processing (where they are input elements). We call this approach "Quasi-3D".

The animation system described here builds on our previous work[4]. Quasi-3D is a 2D cel-based compositing and rendering method using either 2D geometric shapes or raster images as input primitives. The method can deal with cel intersections, so there is complete freedom of cel placement. The output resolution is independently set and, by using a scanline renderer, there is no need to build the image with a conventional frame and z-buffer. Only a scanline is rendered at any given time, reducing memory requirements. Indeed, the output image can be generated directly into a compressed file. High resolution images are thus possible with modest computer resources.

To implement a movie sequence with Quasi-3D, a script file is used to describe the animation. A user can easily manipulate the script by specifying camera, cels, lighting and the paths objects take. The system generates animation frames by calculating the current parameters of the camera and cels, which are then rendered by the quasi-3D method.

The paper is organized as follows. Section 2 introduces the Quasi-3D model and its implementation principles. Section 3 describes the Quasi-3D animation script used to generate multiple frames for an animation and the implementation of the animation system. Section 4 provides two animations generated from the Quasi-3D animation system and Section 5 concludes the paper.

2. The Quasi-3D Model and Rendering

Quasi-3D is a method for animation, composing and rendering using either 2D geometric shapes or raster images as input primitives. It unifies the representation of both movie-quality digital pictures and graphically modelled 2D objects, so that both can be rendered within the same image. The primitives are organized in layers called ‘cels’. Quasi-3D also has basic 3D features such as lighting, perspective projection and 3D movement. A notable feature of Quasi-3D is that it handles hidden-surface elimination (cel intersection) when rendering cels, rather than the simpler layered model of 2D systems. The visible areas of a cel are determined according to its position in a 3D space.

Other effects with Quasi-3D include defocusing, transparency, anti-aliasing, Boolean operations and transformations of primitives. In Quasi-3D, the resolution of the final image is virtually unlimited. As no intermediate frame buffer is used, performance is much less dependent on resolution than with standard programs. This allows rendering very large images in a reasonable time. The Quasi-3D model is described in detail below.

2.1. The Quasi-3D Model

An example is used to explain the Quasi-3D model. Figure 1 shows a 3D scene made from four cels: three walls and one panda painting. The cels with 3D location are used for a particular frame and they are stacked on a rostrum as shown in figure 2 which has a camera, the cels and lights.

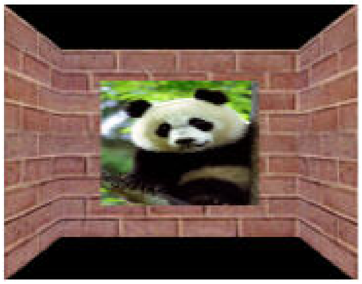

Figure 1: *A Quasi-3D scene using four cels.*

Quasi-3D allows programmers to create virtual stacks of cels as shown in figure 3. The 2D graphics primitives can be transformed and combined using Boolean operators. Lighting can be controlled better than in a real rostrum. Each cel in the stack can have different front and back lights, instead of using a single pair of lights for the stack (see figure 2).

By using the Bath Colour Model[5] we are able to handle transparency and lighting better than with the common alpha channel model.

At each stage of the primitive/object/cel hierarchy, a homogeneous transformation matrix is specified to provide

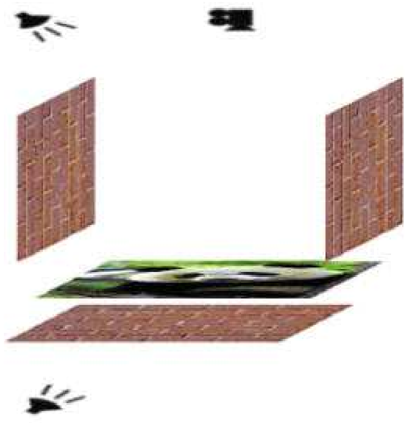

Figure 2: *A rostrum.*

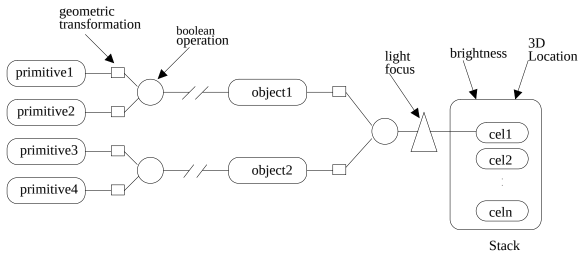

Figure 3: *The stack structure.*

standard operations such as translations, scales, and rotations. In the primitive/object level, the transformation is in 2D and a 3×3 matrix is needed. In the cel level, a 4×4 homogeneous matrix is required for full 3D transformation. More specific operations can be specified at the cel level (e.g. defocus) or at the stack level (e.g. brightness).

A centre of projection and a viewport are used to simulate the 3D placement of the camera. These values can be animated which, in conjunction with the multiple layering, permits panning, zooming and parallax operations.

- Primitives

A primitive is generally defined as a closed region in a 2D space. Rectangles, triangles, circles give simple shapes. Closed NURBS curves provide a general shape. A graphical object can be composed from primitives as shown in figure 3, or from other graphical objects.

- Textures

Each primitive has a texture used to store its colour, independent lighting and other objects. Currently a texture can be one of the following three types:
flat_colour: a single colour is specified for all the points inside the primitive.
gradient: a triangle (containing the primitive) is specified,

as well as the colour of each of its vertices. The colour of a point inside the primitive is then determined by the linear interpolation of these colours.
picture: a raster image is mapped to the primitive's bounding rectangle. This texture type is used to produce digitized images in the composition process.

2.2. Quasi-3D Rendering

Once the above stack structure has been created and the view-related parameters (such as lights and viewport) have been specified, rendering can be performed. Rendering a stack structure is equivalent to shooting a real stack on film: a raster image is computed and then output to one of the available raster devices such as a screen, a film recorder, or an image file.

One of the features of Quasi-3D is handling intersecting cels with arbitrary location and direction in 3D space. To realize perspective projection, 3D space transformations for cels are needed. To eliminate hidden surfaces, cel order is required at each pixel, as shown in figure 4.

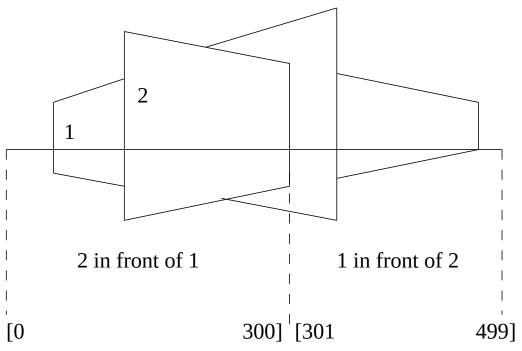

Figure 4: *Cel ordering along a scan-line.*

Rendering and composition are performed scan-line by scan-line. The rendering computes the intersection of primitives with the scan-line and generates a list of colour-coherent spans, which can be single colour, gradient colour or array colour. The composition merges the results from the rendering. The resulting list of spans is finally converted to pixels.

Below is an overview of the rendering process:

- Initialization for rendering, including calculating of primitives' 3D global transformation and computing each cel's supporting plane equation.
- For each scan-line
  - Compute cel order spans for the scan-line. This determines the cel visibility changes along the scan-line (i.e. whether a cel that was in front of another cel "moves" behind somewhere on the scan-line). The cel order information is stored in a list of spans with the corresponding orders. In figure 4, the list will be in the form of: span 1 [0 to 300] order:(1,2), span 2 [301 to 499] order:(2,1).
  - Compute cel colour spans for the scan-line.
  - For each cel order span, composite cels in the corresponding order.

In Quasi-3D rendering, antialiasing is performed by super-sampling and filtering. The user may supply any digital filter, though box and Gaussian functions are built-in. For details about our compositing and filtering see Froumentin [4].

For Quasi-3D, no frame buffer is required as scan-line based computing allows images to be rendered with almost no limit on resolution or aspect ratio. Moreover, because rendering and composition are combined together, there is no need to store rendered components for later composition, saving on both time and storage. Furthermore, Quasi-3D only renders the part of a cel which is within the viewport. This avoids unnecessary computation of hidden objects and thereby improves the overall rendering speed.

3. Quasi-3D Animation

What we have described so far relates to producing a single frame. In order to produce a movie sequence, we need to be able to choreograph the movements of the cels, the movement of the camera and the variation in the lighting. We have implemented a scripting system for this.

3.1. An Animation Script

The animation script file provides controls over graphical objects, virtual cameras and light resources. The script is organized in a structured, consistent and succinct way to facilitate an animator intuitively specifying an animation. Figure 5 shows a sample. There are five parts in the script, SCENE, PATH, CAMERA, LIGHT and SCRIPT. Each part is described below.

- SCENE

The scene of a Quasi-3D animation is composed from 2D cels. The SCENE in the script shows the cel names used to render frames for an animation.

- PATH

In a Quasi-3D animation, both camera and cel can move from one position to another position in a 3D space, with or without direction changes, defined by a PATH. A PATH in the script specifies the starting location and the ending location of camera or a cel. In the sample script, there are three paths, P1, P2 and P3.

- CAMERA

A CAMERA in the script specifies a camera's zoom setting (focal length) and focus setting. These can vary from frame to frame so there are starting and ending values in each case. There are two cameras in the sample script, C1 and C2.

```
SCENE cel1 cel2
PATH
P1
{
(Px1s, Py1s, Pz1s)(Dx1s, Dy1s, Dz1s)Up1s
(Px1e, Py1e, Pz1e)(Dx1e, Dy1e, Dz1e)Up1e
}
P2
{
(Px2s, Py2s, Pz2s)(Dx2s, Dy2s, Dz2s)Up2s
(Px2e, Py2e, Pz2e)(Dx2e, Dy2e, Dz2e)Up2e
}
P3
{
(Px3s, Py3s, Pz3s)(Dx3s, Dy3s, Dz3s)Up3s
(Px3e, Py3e, Pz3e)(Dx3e, Dy3e, Dz3e)Up3e
}
CAMERA
C1
{
CL1s,CF1s
CL1e,CF1e
}
C2
{
CL2s,CF2s
CL2e,CF2e
}
LIGHT
L1
{
(fr1s,fg1s,fb1s)(br1s,bg1s,bb1s)
(fr1e,fg1e,fb1e)(br1e,bg1e,bb1e)
}
L2
{
(fr2s,fg2s,fb2s)(br2s,bg2s,bb2s)
(fr2e,fg2e,fb2e)(br2e,bg2e,bb2e)
}
SCRIPT
cel1 1~200 P1 L1 IN 20 OUT 20
camera 1~60 P2 C1
cel2 5~10 P3 L2
camera 61~200 P2 C2
```

Figure 5: *A Quasi-3D animation script sample.*

- LIGHT

Cel lighting can be changed during an animation. The LIGHT defines the alteration of a cel's front and back lights, represented as (r,g,b). L1 and L2 are two lights in the sample script.

- SCRIPT

The SCRIPT defines all the events during the animation. The events are described in the format:

<actor><startframe∼endframe><variations>

The *actor* can be a camera or a cel. The *startframe* and *endframe* are the indices of the first and last frame to which this action applies.The *variations* define the changes within the range of the frames and the method of interpolation can be linear or slowin/slowout. As an example, consider the line "camera 1∼60 P2 C1" in the sample script. This line specifies that a camera acts from frame 1 to frame 60 with path defined in path P2 and camera parameters defined in C1. "cel1 1∼200 P1 L1 IN 20 OUT 20" means that cel1 acts from frame 1 to frame 200 with path P1 and lighting L1. The slowin/slowout interpolation method is applied to the starting 20 frames and the ending 20 frames respectively.

3.2. Animation Implementation

Using the above script file, the Quasi-3D animation system generates a stack for each frame using the current cels and cameras. The calculation of a stack involves the in-between interpolation and the transformations of cels into the camera coordinate system. The approach used to implement slowin/slowout is parabolic interpolation (constant acceleration) [6]. Once the current frame stack is generated, then a frame can be rendered. Figure 6 shows the Quasi-3D animation generation dataflow.

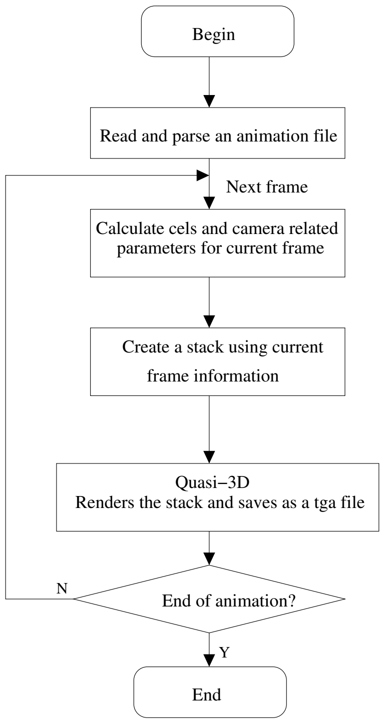
Begin
Read and parse an animation file
Next frame
Calculate cels and camera related parameters for current frame
Create a stack using current frame information
Quasi−3D Renders the stack and saves as a tga file
End of animation?
N
Y
End

Figure 6: *Quasi-3D animation dataflow.*

4. Animation Applications

Computer animations have been widely used in advertising, film special effects and many VR applications. The Quasi-3D animation system can be used to generate animations for the movements of scenes (cel transformations) and camera. In addition, some effects can be achieved by adjusting the camera's focal length, focus and the cel's lighting. We can also generate a 'look at' animation effect by defining two paths for the camera and the cel in which the starting/ending direction vector in the camera path is the same as the starting/ending position vector in the cel path.

Two animations generated by the Quasi-3D animatio system are described below.

4.1. Walking Around Animation

In this first example, we simulate the effect of walking pa: a garden scene, constructed to demonstrate a parallax e: fect. The scene includes four cels: background trees, gras: covered ground, a modern stone sculpture and a classic: statue. During the animation, the cels are fixed in the 3l space and the viewer (viewpoint) moves to the left whil looking into the scene. The script for the animation is give in appendix A. The animation involves 180 frames of whic four frames are shown in figure 7.

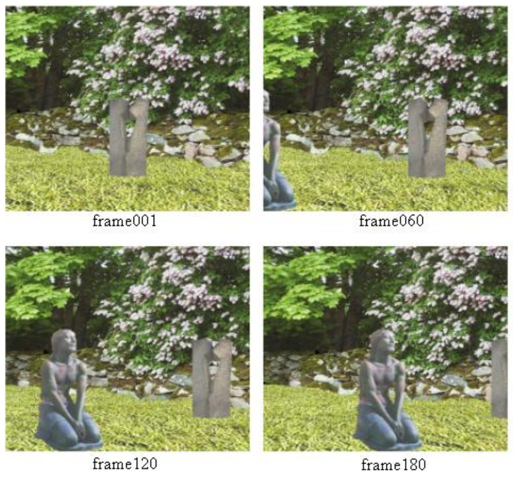

Figure 7: *Four frames in the garden walking around animation.*

The animation shows how the view changes during a walking around. The four frames show the co-locations between the objects and the transparent backgrounds of the sculpture and statue cels. All four cels remain correctly "locked" in 3D position as this proceeds.

4.2. Dynamic Scene Animation

In addition to changing a viewpoint in an animation, we can also change the scene dynamically. To illustrate this, we modelled an object flying towards the camera over a fixed scenic background. In figure 8, a balloon flies above a lake area and approaches the viewer. At the same time the background becomes darker. This is done by specifying the movement of the cel containing the balloon and adjusting the lighting. The animation includes only two cels, one for the background and one for the balloon. The script for the animation is given in appendix B. The balloon animation involves 150 frames of which four frames are shown.

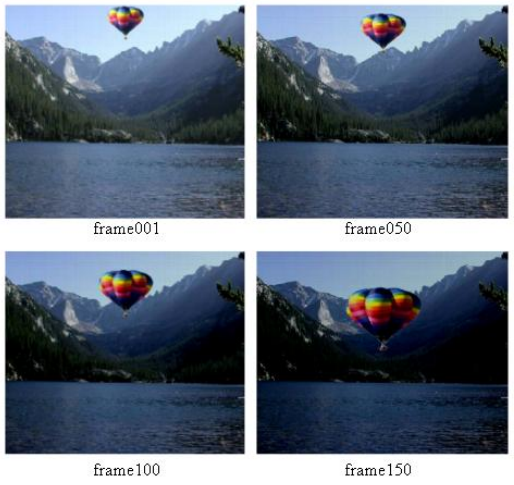

Figure 8: *Four frames in the balloon animation.*

5. Conclusions

In this paper, a Quasi-3D animation system is described. The system consists of an image-based rendering component coupled to a script-based animation component.

Using the system, various 3D effects, lighting and transparency effects can be realized. Quasi-3D is able to render very-high resolution frames by virtue of its scanline approach. The animation script is used to define actors and the timing of events. For simple scenes, the script is manipulated directly. For more complex scenes it would be easy to generate automatically from a front-end application. The two sample animations generated from the system show the simplicity of the Quasi-3D method. A wide range of other operations is supported.

However, there is still some work to be done to improve the Quasi-3D animation system. For instance, any other 2D primitive can be added by providing a procedure for rendering the intersection of that primitive and a scan-line. We have work in hand to include continuous interpolated images[7]. In the animation script, the interpolation method can be improved to offer other forms of interpolation.

Acknowledgements

The Quasi-3D animation system is built on earlier work at MTRC, by previous research staff Max Froumentin, Frederic Labrosse and Sam Laidler. The authors would like to thank them for their contributions. We also thank Dan Su for discussions on the system implementation. We are grateful to the UK EPSRC for funding the Quasi-3D project which employs Dr Qi.

References

1. J. Lengyel and J. Snyder. Rendering with coherent layers. *ACM Computer Graphics (Proc. of SIGGRAPH '97)*, pp. 233–242, 1997.
2. Jay Torborg and James T. Kajiya. Talisman: Commodity realtime3D graphics for the PC. *ACM Computer Graphics (Proc. of SIGGRAPH '96)*, pp. 353–363, 1996.
3. Jean-Daniel Fekete, Érick Bizouarn, Éric Cournarie, Thierry Galas, and Frédéric Taillefer. TicTacToon: A paperless system for professional 2-D animation. *ACM Computer Graphics (Proc. of SIGGRAPH '95)*, pp. 79–90, 1995.
4. M. Froumentin, P.J. Willis. An efficient 2.5D rendering and compositing system. *Computer Graphics Forum (Eurographics'99 Proc.)*, pp. C385–C394, 1999.
5. Robert J. Oddy and Philip J. Willis. A physically based colour model. *Computer Graphics Forum*, **10**(2):121-127, June 1991.
6. Richard Parent. Computer Animation:Algorithms and Techniques. Morgan Kaufmann Publishers, 2001.
7. Dan Su and Philip J. Willis. Demosaicing of colour images using pixel level data-dependent triangulation. acceptted for publication in *EG UK 2003*.

- Appendix A

```
SCENE grassland,backgroundtrees,stone,statue
PATH
P1
{
(0,0,-1)(0,0,1)0
(-1.5,0,-1)(0,0,1)0
}
P2
{
(0,-1,1.9)(0,-1,0)0
(0,-1,1.9)(0,-1,0)0
}
P3
{
(0,0,8)(0,0,1)0
(0,0,8)(0,0,1)0
}
P4
{
(0,-0.4,2)(0,0,1)0
(0,-0.4,2)(0,0,1)0
}
P5
{
(-1.5,-0.4,1)(0,0,1)0
(-1.5,-0.4,1)(0,0,1)0
}
CAMERA
C1
{
1,0
1,0
}
LIGHT
L1
{
(1,1,1)(0,0,0)
(1,1,1)(0,0,0)
}
SCRIPT
camera 1∼180 P1 C1
grassland 1∼180 P2 L1
backgroundtrees 1∼180 P3 L1
stone 1∼180 P4 L1
statue 1∼180 P5 L1
```

- Appendix B

```
SCENE lake,balloon
PATH
P1
{
(0,0,-1)(0,0,1)0
(0,0,-1)(0,0,1)0
}
P2
{
(0,0,12)(0,0,-1)0
(0,0,12)(0,0,-1)0
}
P3
{
(0,5,11)(0,0,-1)0
(0,1,5)(0,0,-1)0
}
CAMERA
C1
{
1,0
1,0
}
LIGHT
L1
{
(1,1,1)(0,0,0)
(0.5,0.5,0.5)(0,0,0)
}
SCRIPT
camera 1∼150 P1 C1
lake 1∼150 P2 L1
balloon 1∼150 P3 L1
```