A Mediated Reality Environment using a Loose and Sketchy rendering technique

Michael Haller, Florian Landerl
Media Technology and Design
Upper Austria University of Applied Sciences
4232 Hagenberg, Austria
name.surname@fh-hagenberg.at

Abstract

In this poster, we present sketchy-ar-us, a modified, real-time version of the Loose and Sketchy algorithm used to render graphics in an AR environment. The primary challenge was to modify the original algorithm to produce a NPR effect at interactive frame rate. Our algorithm renders moderately complex scenes at multiple frames per second. Equipped with a handheld visor, visitors can see the real environment overlaid with virtual objects with both the real and virtual content rendered in a non-photorealistic style.

1. Motivation

In recent years, non-photorealistic rendering (NPR) has become a popular research topic in the area of computer graphics. Our work is based on elements of Mediated Reality, as introduced by Steve Mann. In contrast to Augmented Reality, in a Mediated Reality interface virtual content is not just added to the real environment, but modified by a visual filter. In this poster, we present sketchy-ar-us (cf. figure 1), a modified, real-time version of the Loose and Sketchy algorithm presented by Curtis [1] used to render an AR environment. Curtis' original algorithm was primarily designed for offline rendering, where the performance played a secondary role (it took 10-60 seconds to render each frame). Our Loose and Sketchy algorithm consists of three different steps: finding the silhouette by using reference images, blurring the image to achieve a more fuzzy image, and adding paper texture to produce a more stylistic image. Fischer et al. [3] postulate a cartoon-like AR environment, which is based on a bilateral image filtering for the color segmentation and a Canny-edge-detector for the silhouette generation. We used depth information to create silhouettes of the virtual 3d objects. Thus, 3d objects closer to the camera viewpoint can be drawn with thicker silhouette lines than objects that are far away. Our algorithm is based on a particle system, which allows high flexibility and results in a more stylistic image. Moreover, the silhouette strokes are placed randomly, which itself results in a more ”dynamic” image.

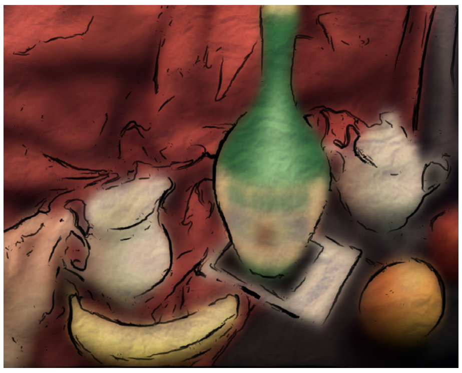
Figure 1. sketchy-ar-us in action.

2. Real-time Loose and Sketchy approach

All the steps involved in creating a Loose and Sketchy rendering are presented below (and shown visually in figure 2):

```
for each frame of animation
  render scene to PBuffer
  generate reference image from depth-buffer
  update particle system with reference image
  blur color-buffer texture

  render blurred color texture to framebuffer
  multiply framebuffer by paper texture
  for each particle (that starts a stroke)
    add brush stroke to framebuffer
  end for
end for
```

Our modified sketchy algorithm can be divided into the following steps:

1. Before we can start to render the scene, we need a **preprocessing** step. Direct rendering to textures using *PBuffers* allows for rapid generaton of the neces-

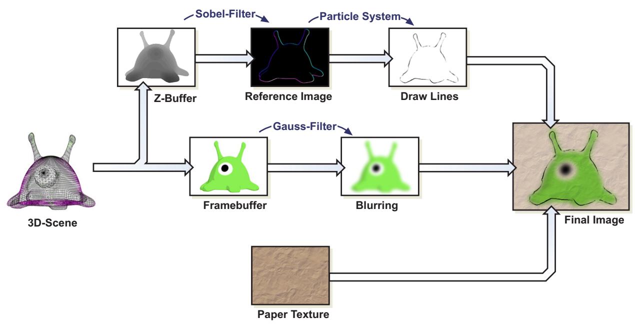

Figure 2. The pipeline of the real-time Loose and Sketchy algorithm.

sary data. Once defined as a render target, the rendering process draws directly into the appropriate color- and depth-buffer textures of the *PBuffer*.

2. Next, a **reference image** has to be created. It contains the silhouette information of the scene as well as a ”force field” which is used to place the strokes along the silhouettes. By applying the Sobel edge detection filter on the depth-buffer texture, the necessary data can be found for constructing the reference image.

3. The reference image shows where the **objects' silhouettes** are to be found. After calculating the reference image using a fragment shader, it is stored as a texture. Since the reference image has to be accessible when drawing the strokes, the contents of this texture have to be read back into main memory. For rendering the brush strokes on the screen we implemented a special particle system. A single stroke consists of multiple particles, which determine the stroke's position and its course. For each frame a predefined number of particles are consecutively emitted and placed randomly on the screen. By looking up the reference image, we assert if the particle's position is located at a silhouette edge and a brush stroke texture is applied accordingly. To guarantee that this does not result in ”stretching artifacts”, the brush stroke textures have to be chosen carefully. Since the strokes are created randomly, they are different every time and a frame-to-frame coherence between the brush strokes is not guaranteed.

4. The blurring is accomplished in hardware using a two-dimensional image-processing filter as described in [2, Cha. 21]. First, the color-buffer texture is blurred along one axis to produce a temporary image. This image is then blurred along the other axis to produce the final blur. As mentioned in the pre-processing step, a texture was prepared containing the color-buffer of the rendered scene. By blurring this texture, the colors appear to fill the strokes completely despite the fact that the strokes do not coincide with the edges of the objects.

To be able to apply the brush strokes that are used for rendering the 3D objects' silhouettes to the background image as well, a reference image for the real environment has to be generated. By applying the same Sobel edge detection filter, albeit with higher threshold settings to the red channel of the background image, we are able to create decent silhouette information for the real environment.

3. Results

By using modern graphics hardware and high-level shader languages, the process is fast enough to be applied in real-time AR environments. By using a 2.8 GHz PC with 1 GB of memory and an nVIDIA Geforce FX 6800 with 256 MB, we achieve 16.87 fps using a geometry complexity of 10,182 polygons.

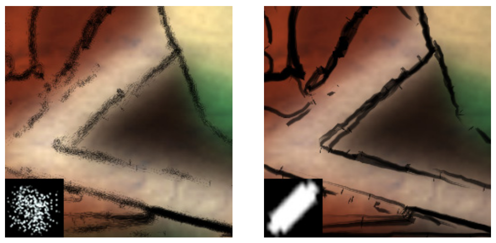

Figure 3. Close-up results of sketchy-ar-us.

One problem of our proposed algorithm is the lag of stylized silhouettes with a robust frame-to-frame coherence. Thus, the ”flickering” effect, as it was proposed by Curtis, can become disturbing once the objects are moving fast. The particle position, which is calculated randomly within the silhouette path, has to be chosen more carefully from one frame to the other. One possible solution would be propagating the parametrization from strokes in one frame to strokes in the next. We would like to combine these presented ideas with our algorithm to achieve more compelling results.

References

[1] C. Curtis. Non-photorealistic animation. In *Proc. ACM SIGGRAPH 1999*. ACM Press, 1999.

[2] R. Fernando. *GPU Gems*. Addison-Wesley, 2004.

[3] J. Fischer, D. Bartz, and W. Straßer. Stylized Augmented Reality for Improved Immersion. In *Proceedings of IEEE Virtual Reality (VR 2005)*, 2005.