9.2

Interactive Image Segmentation Based on GPU Cellular Automata

Frank Nielsen, **Sony Computer Science Laboratories, Inc.**

Introduction

Image segmentation consists of grouping pixels into homogeneous regions representing perceptual units using local and/or global cues. Although the first segmentation algorithms were introduced in the early 1970s, segmentation is still a hot topic of computer vision. Segmentation can be handled at different image understanding scales [Nielsen05]: (1) *low-level* segmentation uses local pixel neighborhood cues such as color or texture information to infer the global image partition; (2) *middle-level* segmentation considers elements of the Gestalt theory such as symmetry detection and rules thereof to improve the pixel grouping; (3) *high-level* segmentation relies on (re)cognition to improve the overall global segmentation (e.g., recognizing first categories of objects lets us later refine their segmentations). Foreground-background segmentation is a simplified segmentation task that seeks to decompose the image into two planes: the *foreground mask plane* and the *background mask plane*. Finally, we distinguish between *hard* and *soft* segmentation, which depends on whether masks have only 0/1 binary values or potentially float values (by analogy to hard/soft clustering). It is well known that segmentation algorithms are time-consuming and that they may work or simply fail, depending on input images. Thus, it is crucial to allow for a rectification mechanism by letting the user *steer* the segmentation using prior cues [Nock05].

To reduce segmentation computation times, the GPU has already been successfully used in the past for fully automatic segmentation, using for example level sets [Lefohn05]. Here, we present a GPU shader implementation for a recent simple yet efficient cellular automata-based foreground/background hard segmentation: GrowCut [Vezhnevets05]. Figure 9.2.1 displays the compositing result obtained after segmenting of a stuffed furry dog image by a GrowCut shader.

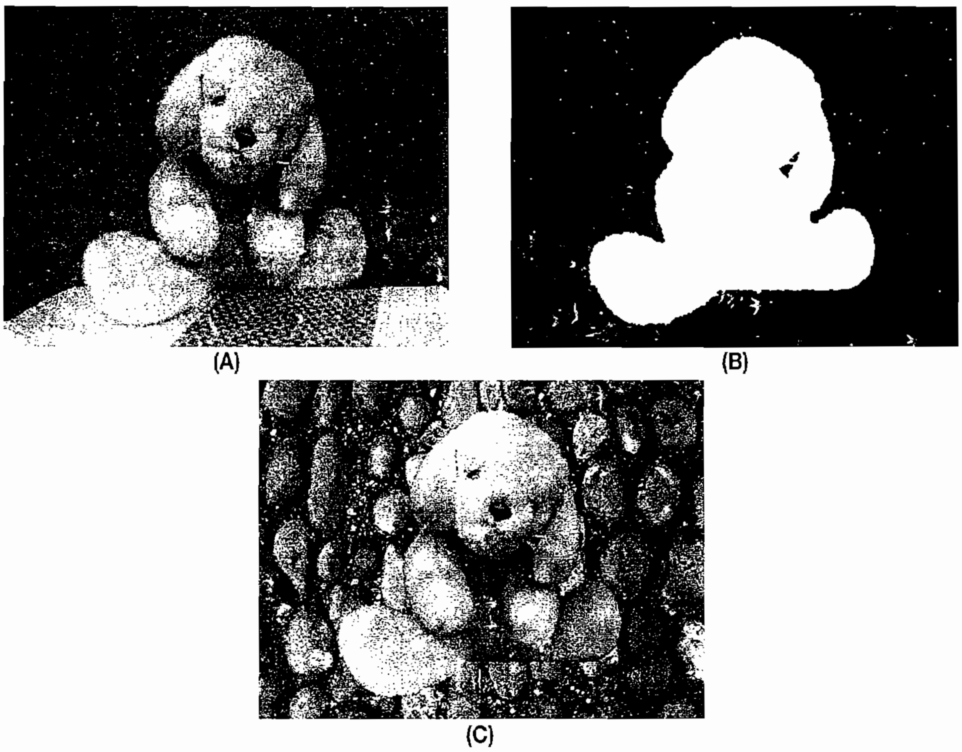

FIGURE 9.2.1 Source image of a (**A**) stuffed furry dog (**B**) with its binary foreground mask obtained from GrowCut. Image (**C**) displays the result of compositing the dog mask with a new background image.

The GrowCut Cellular Automaton Principle

Vezhnevets and Konouchine [Vezhnevets05] recently proposed an elegant and very simple segmentation algorithm, called GrowCut, based on cellular automata. GrowCut performs well in practice: A Photoshop® plug-in that can also be used with shareware or freeware software such as Xnview (*xnview.org*), IrfanView (*irfanview.com*), or Paintshop (*jasc.com*) is available [Vezhnevets05]. GrowCut first initializes a cellular automaton (CA) where each cell associated to an image pixel stores its *state* in a three-tuple (l, s, C). Parameter l denotes the label of cell, namely, the foreground ($l = 1$) or background label ($l = 0$) of the underlying cell's pixel. Scalar variable s denotes the strength of the cell, which is related to the stability of the label. Finally, vector C encodes the color information of the corresponding pixel: A triple of RGB colors. To initialize a GrowCut CA, we set for each pixel p its corresponding CA cell as: $l[p] = 0, s[p] = 0, \mathrm{C}[p] = (R_p, G_p, B_p)$. Then, we require the user to *initially* input foreground and background priors using mouse strokes, as depicted in Figure 9.2.2. We set the corresponding cell states using the stroke label and enforce the initial hard constraints using the maximum strength $s = 1$. Pixel regions not covered by any stroke are labeled unknown ($l = 0.5$) with zero strength. Thus, pixels are initially annotated as being either foreground, background, or unknown (initial trimap). A more flexible initialization will provide a soft background/foreground brush where the strength values may decrease as we near the stroke borders (values within the unit interval).

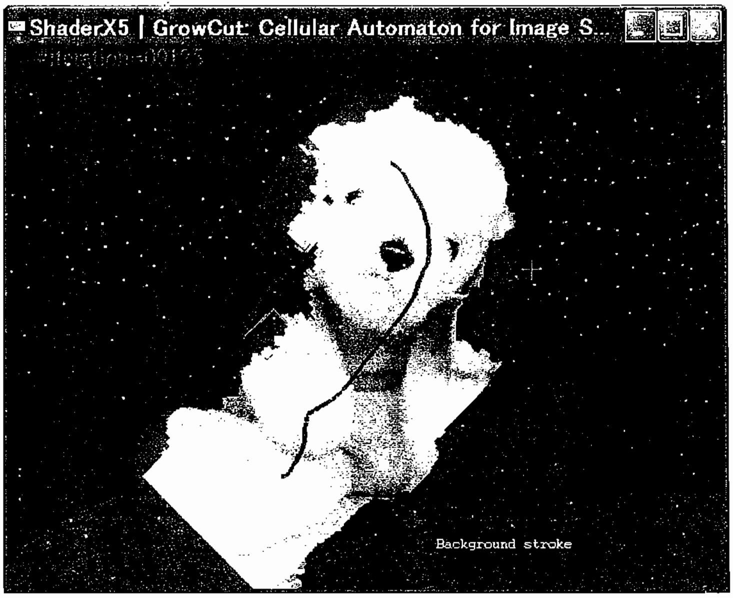

FIGURE 9.2.2 Snapshot of the CPU GrowCut application on a stuffed furry dog image showing the evolution of pixel labels at the 125th iteration.

ON THE CD

Once this initialization step is performed, we let the GrowCut CA evolve on the trimap until it converges into a stable state at which none of the cell states change. Please refer to subfolders, CPU GrowCut UI and CPU Image Composite, in the article's folder on the CD-ROM. The evolution rules are inspired by bacteria behaviors. Bacteria may spread if they can successfully attack some of their neighbors, or conversely, they shrink and potentially vanish if they have been killed by other families of bacteria. Bacteria of a same family are indexed by their common label l. We summarize in pseudo-code the GrowCut CA evolution rule as originally proposed in [Vezhnevets05]:

```
// Pseudo-code for GrowCut CA evolution rules
For all cells
    // Copy the previous state (colors do not change)
```

```
 l'[p] = l[p], s'[p] = s[p]
For all C4 or C8 neighbor q of the current cell
if g(Dist(C[p],C[q])s[q]>s[p] then
    // q successfully attacked p
    // update the cell state
    ;'[p] = l[q],s'[q] = g()Dist(C[p],C[q],C[q]s[q]
```

`Dist()` returns the distance between two RGB colors, and function `g()` is a *monotonous decreasing* function bounded to the unit interval [0,1] that guarantees convergence. For example, we can choose $g(x)=1-\frac{x}{MaxDist}\in[0,1]$.

Choosing the 3D Euclidean distance for comparing (RGB) triples yields $MaxDist = 255\sqrt{3}$ (approximately 441.67). (CIE LAB color space may yield better results, as the Euclidean distance makes sense but would require an extra image conversion step.)

Cellular automata bring to segmentation a *fully dynamic* aspect of region labeling, where pixels' labels may oscillate a few times before reaching their final states. Moreover, at any time, users can further interactively input foreground/background strokes or edit previous strokes to gear the overall segmentation in difficult areas. A naïve CPU implementation (`GrowCutUI`) using a thread for computing the GrowCut cellular automaton yields a low refresh rate. CA, coupled map lattice (CML), or Petri nets are well known to be easily ported to GPU architectures [Harris03] [Lefohn05] (e.g., Conway's game of life). We describe in the next section the GrowCut shader and its performance compared with the nonoptimized CPU version.

A GPU GrowCut Shader

We implemented a simple shader, `growcut.cg`, that examines in a deterministic order the C4 (four-connectivity of pixels) neighbors of a pixel to decide whether to relabel its neighbor. Because computations are carried out in parallel between any two successive iterations, this amounts to deciding for each pixel whether it keeps its label, and updating its strength accordingly. Our shader is a direct translation of the basic C++ loop code. In this implementation we deterministically examine the neighborhood of pixels and store the five parameters (*R, G, B, l, s*) into two texture units: the source color image (*R, G, B*) and the label-strength image (*l, s*). Considering a random order of inspection of a pixel's neighborhood will yield a better segmentation but remove privileged-direction-growth artifacts. The shader in the following listing and its accompanying program can further be improved in several ways such as taking into account several objects instead of plain foreground/background segmentation (multi-label), handling 3D volumetric image stacks, and smoothing the object boundaries using a somewhat more complex evolution rule [Vezhnevets05].

```
float Dist(float3 p, float3 q)
{return sqrt((p[0]-q[0])*(p[0]-q[0])+(p[1]-q[1])*
(p[1]-q[1])+(p[2]-q[2])*(p[2]-q[2]));}

sqrt(3) is approximately 1.7321
float g(float x)
{return 1.0-x/(1.7321);}

void GrowCut( float2 texCoord:TEXCOORD0, out float3 color: COLOR,
uniform samplerRECT Image, uniform samplerRECT Label)
{
// Neighborhood in color RGB image
float3 pixel = f3texRECT(Image, texCoord);
float3 neighUp = f3texRECT(Image, texCoord+float2(0,1)) ;
float3 neighDown = f3texRECT(Image, texCoord+float2(0, 1) ) ;
float3 neighLeft = f3texRECT(Image, texCoord+float2(-1,0) ) ;
float3 neighRight = f3texRECT(Image, texCoord+float2(1,0) ) ;

// Neighborhood in (Label,Strength) image
float3 pixell = f3texRECT(Label, texCoord);
float3 neighUpl = f3texRECT(Label, texCoord+float2(0,1)) ;
float3 neighDownl = f3texRECT(Label,  texCoord+float2(0,-1) ) ;
float3 neighLeftl = f3texRECT(Label, texCoord+float2(-1,0) ) ;
float3 neighRightl = f3texRECT(Label, texCoord+float2(1,0) ) ;

// Initialization
float label=pixell[0];
float strength=pixell[1];

float pstrengthUp=neighUpl[1]*g(Dist(neighUp[0],pixel[0]));
float pstrengthDown=neighDownl[1]*g(Dist(neighDown[0],pixel[0]));
float pstrengthLeft=neighLeftl[1]*g(Dist(neighLeft[0],pixel[0]));
float pstrengthRight=neighRightl[1]*g(Dist(neighRight[0],pixel[0]));

float diffUp=pstrengthUp-strength;
float diffDown=pstrengthDown-strength;
float diffLeft=pstrengthLeft-strength;
float diffRight=pstrengthRight-strength;

//
//Neighbor UP tries to attack me
//
    if ((neighUpl[0]!=pixell[0])&&(diffUp>0.0))
        {
        // Neighbor Up succeeded the attack
        label=neighUpl[0];
        strength=pstrengthUp;
        }
        else
        {
```

```
                    // Neighbor DOWN tries to attack me
                    if ((neighDownl[0]!=pixell[0])&&(diffDown>0.0))
                    {
                    label=neighDownl[0];
                    strength=pstrengthDown;
                    }
                    else
                        { // Neighbor LEFT tries to attack me
                            if ((neighLeftl[0]!=pixell[0])&&(diffLeft>0.0))
                            {
                            label=neighLeftl[0];
                            strength=pstrengthLeft;
                            }
                            else
                            { // Neighbor RIGHT tries to attack me
                            if ((neighRightl[0]!=pixell[0])&&
                            (diffRight>0.0))
                            {
                            label=neighRightl[0];
                            strength=pstrengthRight;
                            }
                        }
                    }
            }
        // New LS state for the ImageLS
        color=float3(label,strength,0);
        }
```

For pedagogical reasons, we chose to implement a screen-frame buffer rendering that shows the evolution of the CA state at each iteration. A better-optimized implementation would consider off-screen rendering using common GPGPU techniques: render to pbuffer, render to texture, or use a frame buffer object (OpenGL FBO extension GL_FRAME_BUFFER_EXT [FBO05]).

Finally, we export the foreground mask using the two texture images (ImageRGB, ImageLS) and using another fragment shader, GrowCutExport (see Figure 9.2.3):

```
void  GrowCutExport(float2 texCoord:TEXCOORD0, out float3 color:
COLOR, uniform samplerRECT ImageRGB, uniform samplerRECT ImageLS)
{
float3 pixellabel = f3texRECT(ImageLS, texCoord);

// extract only the foreground object from the image
if (pixellabel[0]==1.0)
        {
        color=f3texRECT(ImageRGB, texCoord);
        }
        else
        {// background color
        color=float3(0.1,0.8,0.1);
}
}
```

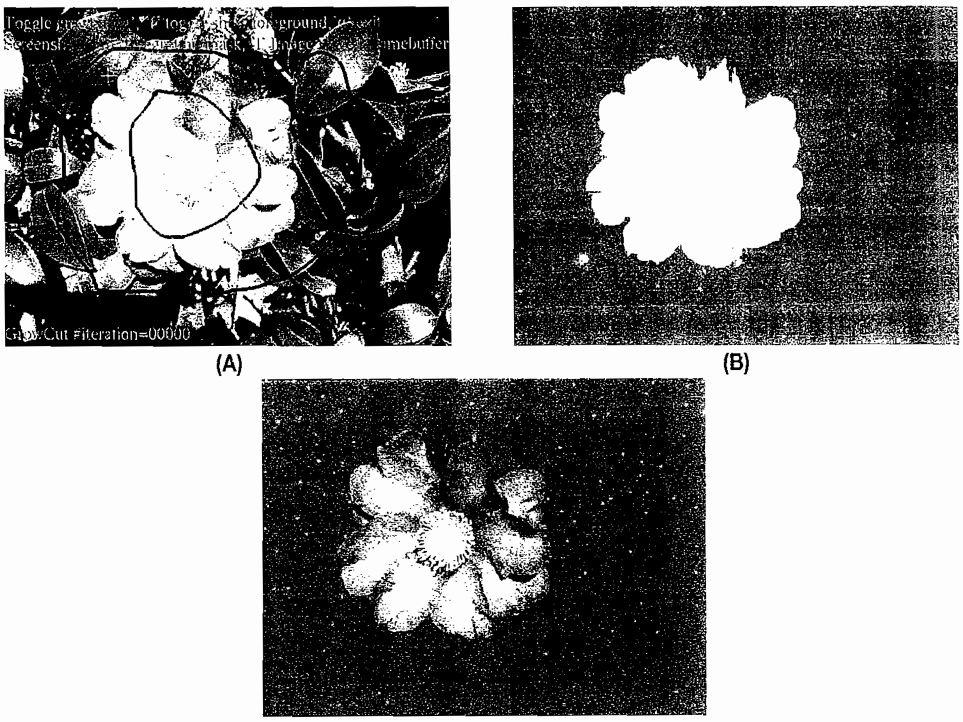

FIGURE 9.2.3 Snapshot of the GPU GrowCut application (**A**) on a flower image initialized interactively using a foreground/background stroke, (**B**) on a label-strength image after 300 automaton iterations, and (**C**) on an extracted foreground flower.

Conclusion

We have presented a GPGPU pixel shader for speeding up the simple foreground/background segmentation task using a cellular automaton, which can be found in the article subfolder called GPU GrowCut. The shader implements the simple yet effective GrowCut segmentation method [Vezhnevets05].

ON THE CD

References

[FBO05] OpenGL Framebuffer Extension (FBO). Available online at *http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt.*

[Lefohn05] Lefohn, Aaron, Ian Buck, Patrick McCormick, John Owens, Timothy Purcell, and Robert Strzodka. "General Purpose Computation on Graphics Hardware." *IEEE Visualization 2005* (VIS'05), 2005.

[Harris03] Harris, Mark J. "Real-time Cloud Simulation and Rendering." University of North Carolina, #TR03-040, 2003. Available online at *http://www.markmark.net/dissertation/.*

[Nielsen05] Nielsen, Frank. *Visual Computing: Geometry, Graphics, and Vision.* Charles River Media, 2005. (*http://www.csl.sony.co.jp/person/nielsen*)

[Nock05] Nock, Richard and Frank Nielsen. "Semi-Supervised Statistical Region Refinement for Color Image Segmentation." *Pattern Recognition,* 38(6), (2005): 835–846.

[Vezhnevets05] Vezhnevets, Vladimir and Vadim Konouchine. "Grow-Cut - Interactive Multi-Label N-D Image Segmentation." Graphicon, 2005. Available online at *http://research.graphicon.ru/growcut/gml-growcut.html.*