Computer Graphics

Alexandre Tolstenko Nogueira
InfiniBrains
tolstenko@infinibrains.com.br

1 Atividade Aula 4 - Fundamentos de Processamento Gráfico

Questionário

1)Continuar a implementação do programa de processamento de imagens da aula anterior: implementar o filtro passa-alta e mais dois métodos de detecção de bordas (você pode escolher). Devem ser entregues em um arquivo PDF: o programa fonte e um exemplo de imagem processada (incluir imagem original e imagem resultante) para cada um dos três métodos implementados

Todas as implementações estão disponíveis publicamente no github da game engine que estou desenvolvendo [Tolstenko 2018].

A diferença básica entre o filtro de Sobel e o de kirsch é que o filtro de segundo consegue capturar melhor as nuances de gradientes diagonais.

Apply Mask

```
float EditorGUI::ApplyMask(unsigned char *input, int width, int height, int
    line, int column, float *mask, int maskWidth, int maskHeight, int
    bytesPerChannels, int numberOfChannels) {
  // TODO: handle borders
  if (column >= width - maskWidth/2 + 1 || column <= maskWidth/2 - 1 || line
      >= height - maskHeight/2 + 1 || line <= maskHeight/2 - 1)
    return 0;

  float ret = 0;
  for (int y = -maskHeight/2; y <= maskHeight/2; y++) {
    for (int x = -maskWidth/2; x <= maskWidth/2; x++) {
      int pos = ((line+y) * width + column+x) * numberOfChannels *
          bytesPerChannels;
      float color = 0;
      for (auto i = 0; i < numberOfChannels - 1; i++)
        color += input[pos + i];
      color /= numberOfChannels - 1; // the average of all channels
      color *= mask[x + maskWidth/2 + (y + maskHeight/2) * maskWidth]; //
          apply mask
      ret += color; // accumulate the value
    }
  }

  return ret;
}
```

Laplace

```
void EditorGUI::Laplace(unsigned char * input, unsigned char * output, int
    width, int height, int bytesPerChannels, int numberOfChannels)
{
  if(numberOfChannels!=4 || bytesPerChannels!=1)
    throw NotImplementedException();

  float laplace[] = {-1,-1,-1,
                     -1, 8,-1,
                     -1,-1,-1};

  for(int line =0;line<height;line++){
    for(int column =0;column<width;column++){
      int pos = (line * width + column) * numberOfChannels * bytesPerChannels;

      auto value = ApplyMask(input,width,height,line,column,laplace,3,3);
      auto color = (unsigned char) MIN(MAX(value,0),255);

      output[pos] =color;
      output[pos+1]=color;
      output[pos+2]=color;
      output[pos+3]=255;
    }
  }
}
```

Sobel

```
void EditorGUI::Sobel(unsigned char *input, unsigned char *output, int
    width, int height, int bytesPerChannels, int numberOfChannels)
{
  if(numberOfChannels!=4 || bytesPerChannels!=1)
    throw NotImplementedException();

  float vert[] = {-1,0,+1,
                  -2,0,2,
                  -1,0,1};
  float hor[] = { 1, 2, 1,
                  0, 0, 0,
                 -1,-2,-1};

  for(int line =0;line<height;line++){
    for(int column =0;column<width;column++){
      int pos = (line * width + column) * numberOfChannels * bytesPerChannels;

      auto vertvalue = ApplyMask(input,width,height,line,column,vert,3,3);
      auto horivalue = ApplyMask(input,width,height,line,column,hor,3,3);

      auto color = (unsigned char) MIN(MAX(sqrt(vertvalue*vertvalue +
         horivalue*horivalue),0),255);

      output[pos] =color;
      output[pos+1]=color;
      output[pos+2]=color;
      output[pos+3]=255;
    }
  }
}
```

Kirsch

```
void EditorGUI::Kirsch(unsigned char *input, unsigned char *output, int width, int height, int
    bytesPerChannels, int numberOfChannels)
{
  if(numberOfChannels!=4 || bytesPerChannels!=1)
    throw NotImplementedException();

  float v1[] = { 5, 5, 5,
                -3, 0, -3,
                -3, -3, -3};
  float v2[] = { 5, 5, -3,
                 5, 0, -3,
                -3, -3, -3};
  float v3[] = { 5, -3, -3,
                 5, 0, -3,
                 5, -3, -3};
  float v4[] = {-3, -3, -3,
                 5, 0, -3,
                 5, 5, -3};
  float v5[] = {-3, -3, -3,
                -3, 0, -3,
                 5, 5, 5};
  float v6[] = {-3, -3, -3,
                -3, 0, 5,
                -3, 5, 5};
  float v7[] = {-3, -3, 5,
                -3, 0, 5,
                -3, -3, 5};
  float v8[] = {-3, 5, 5,
                -3, 0, 5,
                -3, -3, -3};

  for(int line =0;line<height;line++){
    for(int column =0;column<width;column++){
      int pos = (line * width + column) * numberOfChannels * bytesPerChannels;

      auto r1 = ApplyMask(input,width,height,line,column,v1,3,3);
      auto r2 = ApplyMask(input,width,height,line,column,v2,3,3);
      auto r3 = ApplyMask(input,width,height,line,column,v3,3,3);
      auto r4 = ApplyMask(input,width,height,line,column,v4,3,3);
      auto r5 = ApplyMask(input,width,height,line,column,v5,3,3);
      auto r6 = ApplyMask(input,width,height,line,column,v6,3,3);
      auto r7 = ApplyMask(input,width,height,line,column,v7,3,3);
      auto r8 = ApplyMask(input,width,height,line,column,v8,3,3);

      // 765 == 3*255 -> this is used to clamp the maximum pixel value properly
      auto max = MIN(MAX(MAX(MAX(MAX(r1,r2),MAX(r3,r4)), MAX(MAX(r5,r6),MAX(r7,r8))),0),765)/3;
      auto color = (unsigned char) max;

      output[pos] =color;
      output[pos+1]=color;
      output[pos+2]=color;
      output[pos+3]=255;
    }
  }
}
```

Figure 1: *Quadrante superior esquerdo: imagem original; superior direito: aplicação do filtro laplaciano; inferior esquerdo: aplicação do filtro de sobel; inferior direito: aplicação do filtro de kirsch.*

2) Encontre as transformações de visualização de um sistema cuja camera está posicionada na posição $(3, 3, 3)$, o plano de projeção é definido pelo ponto $(1, 1, 1)$ e pelo vetor $(-1, -1, -1)$ e o espaço de imagem é definido por um quadrado de lado 20 centrado na origem do plano de projeção.

3) Considere este mesmo modelo de camera e ainda $s_x = s_y = 1$, $u_c = 600$ e $v_c = 400$ e encontre as coordenadas da projeção do ponto $(0, 0.5, 0.5) \in R^3$

4) Descreva como fica a transformação projetiva com mais de um ponto de fuga.

Infelizmente não consegui terminar, mas encontrei um excelente material:

https://www.scratchapixel.com/lessons/3d-basic-rendering/computing-pixel-coordinates-of-3d-point/mathematics-computing-2d-coordinates-of-3d-points

References

TOLSTENKO, A. N., 2018. Mobagen - module based game engine. https://github.com/InfiniBrains/mobagen.