Виртуальная реальность

занятие №10

Рябинин Константин Валентинович

e-mail: icosaeder@ya.ru
jabber: icosaeder@jabber.ru

Пермь, 2012

Фильтрация изображений

Фильтрация изображения – это процесс изменения исходного изображения при помощи некоторой функции (фильтра)

- Вообще говоря, фильтром может называться любая функция, изменяющая изображение
- Фильтр принимает на вход исходное изображение и, возможно, некоторые дополнительные данные, как связанные с изображением, так и не связанные с ним
- Результатом применения фильтра является другое изображение, вообще говоря, несводимое к исходному
- Фильтрация применяется для
  - Повышения качества изображения
  - Достижения специального визуального эффекта
  - Адаптации изображения к условиям конкретной задачи (например, при распознавании образа)

Матрица конволюции

Конволюция – это процесс замены точки изображения на средневзвешенное (с различными весовыми коэффициентами) значение его малой окрестности

Матрица конволюции – это матрица, задающая правило вычисления средневзвешенного значения

Общий алгоритм применения матрицы конволюции:

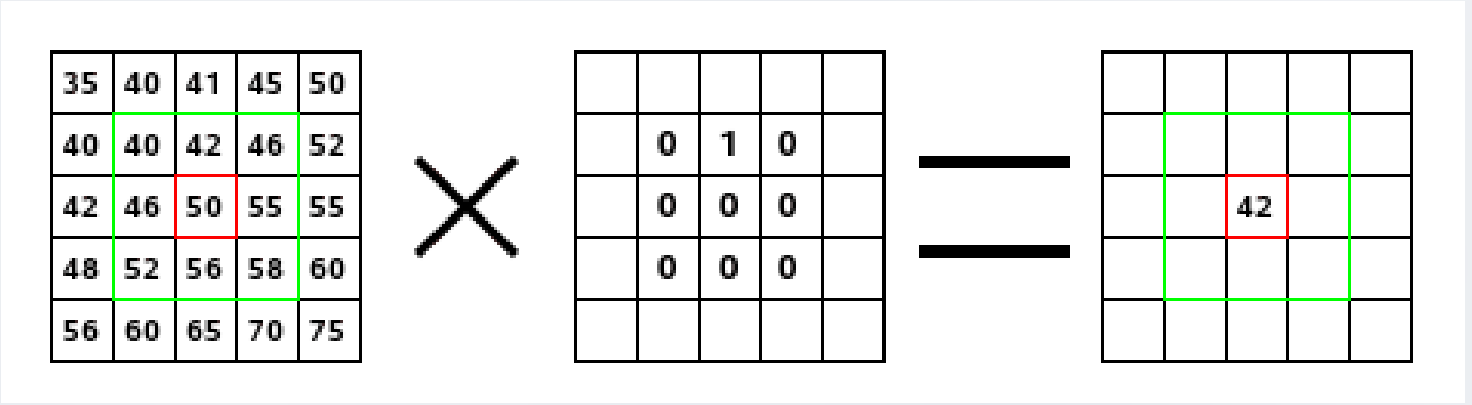
- Пусть есть матрица *m* x *n* (чаще всего *m* = *n*)
- Апертура («окошко») размером *m* x *n* пробегает *своей центральной точкой* по всем точкам изображения
- Цвет каждой точки умножается на соответствующий коэффициент, взятый из матрицы
- Полученные значения складываются и делятся на сумму всех коэффициентов (нормируются). Возможно, к нормированной величине прибавляется некоторая константа
- Полученное значение присваивается точке изображения, которая соответствует центральной точке апертуры

Матрица конволюции

- Конволюция – это наиболее популярный алгоритм фильтрации изображений, так как, обладая простой реализацией, этот способ покрывает огромное количество самых разнообразных эффектов, от коррекции резкости до выделения контуров и рельефа
- Достоинством алгоритма является линейное время фильтрации
- На основе конволюции реализуются и более сложные – многопроходные – фильтры, представляющие собой дифференциальные операторы (дифференциал аппроксимируется свёрткой, получаемой с помощью матрицы конволюции)

Матрица конволюции

- Чаще всего матрица конволюции является квадратной и имеет размерность 3x3 или 5x5



- Во многих графических редакторах (например, в GIMP) матрица конволюции, элементы которой может задавать пользователь, выделена в отдельный фильтр
- Для вывода весовых коэффициентов используется аппарат высшей математики (например, в процессе работы с дифференциалами цвета)

Матрица конволюции

Примеры:

- **Увеличение контраста**

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

- **Равномерное размытие**

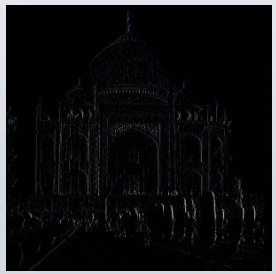
$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

- **Размытие по Гауссу**

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

- **Усиление карёв**

$$\begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$



Матрица конволюции

Примеры:

- **Выделение контура**

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

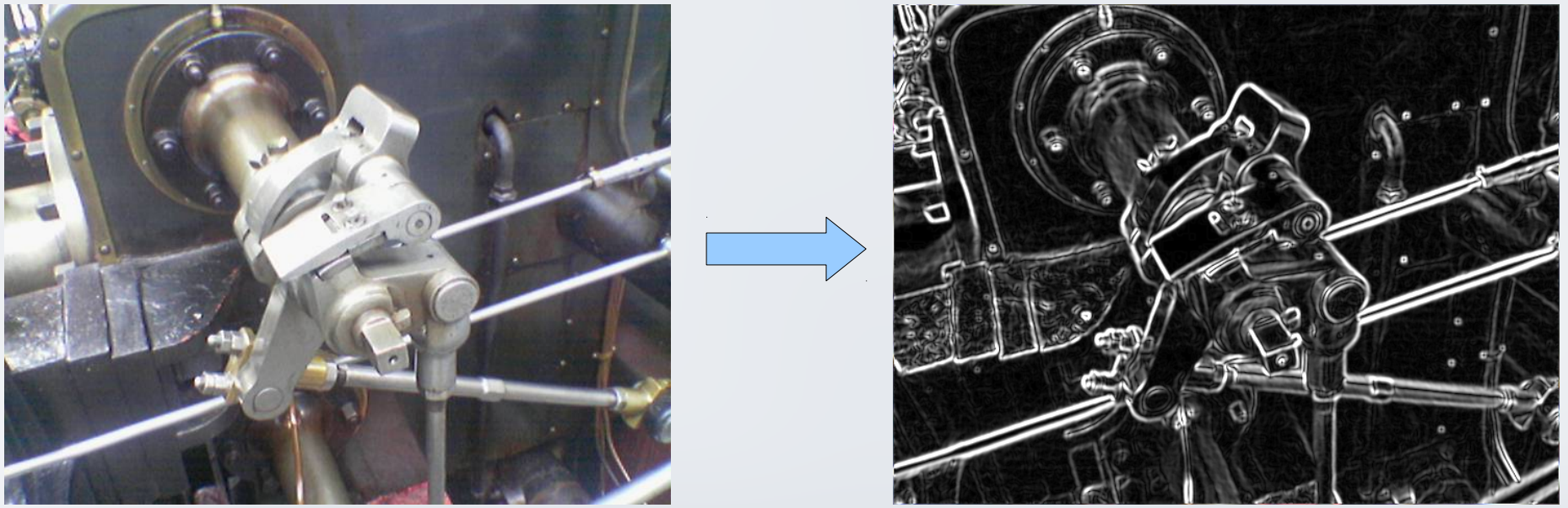
- **Придание рельефа**

$$\begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

Оператор Собела

Более сложный пример – дифференциальный оператор Собела

Фильтр Собела визуализирует градиент яркости, в результате чего на изображении подсвечиваются границы



Оператор Собела

Фильтр Собела требует уже двух проходов с разными матрицами конволюции (для вычисления X и Y координат градиента в каждой точке):

$$G_x = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad G_y = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

$$I' = \sqrt{(G_x[I])^2 + (G_y[I])^2}, \text{ где}$$

I – исходное изображение

I' – результирующее изображение

$G[I]$ – конволюция изображения

Операции возведения в степень выполняются покомпонентно

Конволюция на шейдерах

Конволюция может быть реализована во фрагментном шейдере:

```
uniform sampler2D u_image;
uniform vec2 u_textureSize;
uniform float u_kernel[9];
uniform float u_kernelWeight;

varying vec2 v_texCoord;

void main()
{
   vec2 onePixel = vec2(1.0, 1.0) / u_textureSize;

   vec4 colorSum =
     texture2D(u_image, v_texCoord + onePixel * vec2(-1, -1)) * u_kernel[0] +
     texture2D(u_image, v_texCoord + onePixel * vec2( 0, -1)) * u_kernel[1] +
     texture2D(u_image, v_texCoord + onePixel * vec2( 1, -1)) * u_kernel[2] +
     texture2D(u_image, v_texCoord + onePixel * vec2(-1,  0)) * u_kernel[3] +
     texture2D(u_image, v_texCoord + onePixel * vec2( 0,  0)) * u_kernel[4] +
     texture2D(u_image, v_texCoord + onePixel * vec2( 1,  0)) * u_kernel[5] +
     texture2D(u_image, v_texCoord + onePixel * vec2(-1,  1)) * u_kernel[6] +
     texture2D(u_image, v_texCoord + onePixel * vec2( 0,  1)) * u_kernel[7] +
     texture2D(u_image, v_texCoord + onePixel * vec2( 1,  1)) * u_kernel[8];

   gl_FragColor = vec4((colorSum / u_kernelWeight).rgb, 1.0);
}
```

Конволюция на шейдерах

Передача параметров:

```
...

var kernelLocation = gl.getUniformLocation(program, "u_kernel[0]");
var kernelWeightLocation = gl.getUniformLocation(program, "u_kernelWeight");

...

var edgeDetectKernel = [
    -1, -1, -1,
    -1,  8, -1,
    -1, -1, -1
];
var weight = 0;
for (var i = 0; i < 9; i++)
  weight += edgeDetectKernel[i];
gl.uniform1fv(kernelLocation, edgeDetectKernel);
gl.uniform1f(kernelWeightLocation, weight);

...
```