Министерство образования и науки РФ
ГОУ ВПО Рыбинский государственный авиационно-
технический университет имени П.А. Соловьева

Факультет радиоэлектроники и информатики
Кафедра математического и программного обеспечения
электронных вычислительных средств

Специальность
«Програмное обеспечение вычислительной техники
и автоматизированных систем»

Лабораторная работа №2

по дисциплине
«Компьютерная графика»

Студенты гр. ИВП-09 _____ Кулаевский Д.Ю.
Руководитель _____ Малышев О.В.

Рыбинск 2012

1 Постановка задачи

Разобрать и реализовать алгоритмы уменьшения количества цветов изображения.

2 Карткие теоретические сведения

Старые видеоадаптеры поддерживали только 2, 4, 16 или (позднее) 256 цветов из за ограничений памяти. Приходилось идти на компромис, выбирая между разрешением экрана и количеством доступных цветов. Алгоритмы уменьшения количества цветов помогли, в какой то степени, решить эту проблему, позволив рисовать на экранах многоцветные изображения с использованием ограниченных палитр. Обычно в операционной системе уменьшение количества цветов проводилось автоматически, то есть цвета изображения приводились к палитре операционной системы. Современные видеоадаптеры и мониторы могут отображать миллионы цветов одновременно (при этом сохраняя высокое разрешение), даже больше, чем способен различить человеческий глаз, а это значит, что уменьшения количества цветов операционной системой стало не актуально.

В наше время алгоритмы уменьшения количества цветов применяются при переводах изображения в различные форматы. В основном это форматы PNG и GIF. GIF долго время являлся основным форматов изображений и анимации в Интернет, он поддерживал только 256 цветов. Некоторые ранние браузеры использовали особую палитру, называемую веб-палитра, призванную улучшить качество и стандартизировать палитры.

Эпоха ранних компьютеров продолжалась достаточно долгое время и было придумано большое количество алгоритмов уменьшения количества цветов, позволявших получать на выходе изображения без заметной потери качества.

При уменьшении количества цветов несомненно теряется качество картинки. Чтобы потери качества были как можно менее заметны и применяют специальные алгоритмы.

Самый простой и наиболее известный алгоритм уменьшения количества цветов — это алгоритм порогового уровня или ближайшего цвета, придуманный Паулем Хекбертом в 1980г. (англ. Median cut) который заменяет оригинальный цвет в изображении ближайшим к нему цветом, имеющимся в выбранной палитре. Разница между цветами часто определяется как Эвклидово расстояние в трехмерном цветовом пространстве. Другими словами, если мы имеем цвет (r_1, g_1, b_1) и хотим найти (r_2, g_2, b_2), который соответствует цвету в выходной палитре, то мы должны минимизировать значение:

$$\sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

Error diffusion — это алгоритм уменьшения количества цветов изображения, где квантование каждого пикселя оказывает влияние на процесс квантования соседних пикселей. В отличии от многих других алгоритмов полутонирования в error diffusion отдельные зоны изображения влияют друг на друга, что требует дополнительных буфферов и усложняет распаралелливание. В ходе работы алгоритм заменяет оригинальные цвета ближайшими к ним из выбранной цветовой палитры с учетом полученной ошибки при выборе цветов соседних точек. Алгоритм пробегает изображение слева-направо сверху-вниз, квантуя каждый пиксель поочередно, распространяя ошибку на соседей (кроме пикселей, которые уже кван-

тованы). Флойд и Стейнберг описали реализацию error diffusion для простого ядра

$$\frac{1}{16}\begin{bmatrix} - & \# & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

Также существует множество других реализаций, использующие разные ядра для распространения ошибки. Этот алгоритм относится к группе алгоритмов дизеринга (англ. dithering), которые подразумевают внесения некоторого шума в изображение, для создания эффекта объема, как от использования большого количества цветов. В качестве такого шума в этом алгоритме используется ошибка, распространяемая на соседние пиксели.

3 Исходный код основных процедур программы

3.1 Квантование цвета алгоритмом порогового уровня

```
uchar transformColor(uchar &color,uint x, uint y)
{
    uchar result = 0;

    if(color>127)
        result = 255;

    return result;
}
```

3.2 Квантование цвета алгоритмом случайного порогового уровня

```
uchar transformColor(uchar &color, uint x, uint y)
{
    uchar result = 0,
          random = rand()%255;

    if(color>random)
        result = 255;

    return result;
}
```

3.3 Квантование цвета алгоритмом замены точки матрицей

```
    QColor rgb = QColor(in_img->pixel(x, y));
    uchar color = rgb.red();
    for(int y_p=0; y_p<p_size; ++y_p)
       for(int x_p=0; x_p<p_size; ++x_p)
       {
           int limit = 255/qq;
           if(color>limit*((*pattern)(x_p,y_p)+1))
               r_t->setPixel(x*p_size+x_p, y*p_size+y_p, qRgb
                  (255,255,255));
       }
```

3.4 Квантование цвета алгоритмом Ordered Dither

```
uchar transformColor(uchar &color, uint x, uint y)
{
    int p_size = pattern->xSize(),
        qq = p_size*p_size,
        limit = 255/qq,
        xx = x%p_size, yy = y%p_size;
```

```
    uchar result = 0;

    if(color>limit*((*pattern)(xx,yy)+1))
        result = 255;

    return result;
}
```

3.5 Квантование цвета алгоритмом Error Diffusion

```
uchar transformColor(uchar &color, uint x, uint y)
{
    int div = ((int)(*error)(x, y)/quot),
        sum = ((int)color+div);
    if(sum <0)
        sum = 0;
    if(sum >255)
        sum = 255;
    uchar true_color = sum,
          result = Helper::transformColor(true_color, x, y),
          quot_error = result - true_color;

    for(int xx=pattern->getXBegin(); xx<=pattern->getXEnd(); ++xx)
        for(int yy=pattern->getYBegin(); yy<=pattern->getYEnd(); ++yy)
        {
            if(xx<0 || xx>error->getXEnd() || yy<0 || yy> error->getYEnd
               ())
                continue;
            (*error)(xx+x,yy+y) = (int)(*pattern)(xx,yy)*quot_error;
        }

    return result;
}
```

4 Выводы

В ходе лабораторной работы были изучены различные алгоритмы уменьшения количества цветов, их плюсы и минусы, причину появления и т.д. Большинство алгоритмов уменьшения количества цветов обрабатывают каждый пиксель отдельно, что дает большую возможность для распараллеливания. Однако в алгоритме error diffusion, который дает неплохие результаты, результат обработки каждого пикселя зависит от результата обработки. Алгоритм замены точки матрицей дает неплохой результат, но количество точек на изображении увеличивается в разы. Алгоритм Ordered Dither лишен этого недостатка и также использует матрицы для квантования, но он намного больше зависит от размера используемой матрицы и изображения.