天津大學

图形学实验报告

—— 应用改进的 tone mapping 算法显示高对比度的图片场景

学　　院 计算机科学与技术

专　　业 计算机科学与技术

年　　级 2012

姓　　名 王雨滕

学　　号 3012216083

2014 年 10 月 18 日

一、 实验的内容和目的

✧ 内容：实现论文《Adaptive Logarithmic Mapping For Displaying High Contrast Scenes》中提到的经改进的快速且有效的 tone mapping 技术
✧ 目的：学会使用并深刻理解 tone mapping 算法

二、 实验原理

1. 读入超过 24bit 的 RGB 图像，利用 tone mapping 中的空域不变演算法，建立在快速简单的亮度感知的架构上，通过全局映射的方式，将图片重新映射到对显示实体有较佳效果的显示范围，保留原始的光源信息。

2. 这篇文章提出的的对传统的 tone mapping 方法的改进，是建立在下面的重要公式之上：

$$L_d = \frac{\log(L_w + 1)}{\log(L_{w\max} + 1)}$$

公式中，L_d是映射后的亮度值，L_w是原始图片中将被映射的亮度值，L_{wmax}是原始图片中最大的亮度值。

利用公式中对数的关系的变化，可以简单快速的做到 tone mapping 的效果。但是经过观察得到，在对数中，若利用不同的底数值，会产生不同的结果。如果以 2 为底数，暗部细节比较好，但是太亮；而以 10 为底亮部细节比较好，但是图片太暗。

3. 然而，通过插值算法可以解决上述问题。通过插值让底数在映射过程中，能够在区间[2,10]内变化，从而找到我们需要的结果。

为了得到合适的插值底数，需要用到 bias 力量函数：

$$bias_b(x) = t^{\frac{\log(b)}{\log(0.5)}}$$

其中， 主要参数为 b,不同的 b 也会得到不同的结果。

4. 在本篇文章的算法中上述两个公式结合起来将这样使用：

$$L_d = L_{d\max} \cdot \frac{\log(L_w + 1)}{\log\left(2 + \left(bias\left(\frac{L_w}{L_{w\max}}\right)\right) \cdot 8\right)} \Big/ \log_{10}(L_{w\max} + 1)$$

这个公式是此改进后的 tone mapping 算法的核心公式。算法中利用这个公式对 RGB 转化成 XYZ 之后的 X,Y,Z 值分别进行处理，得到新的 XYZ 值再转化回 RGB 值，将矩阵写入.bmp 文件中，最后从位图文件中提取这个矩阵。

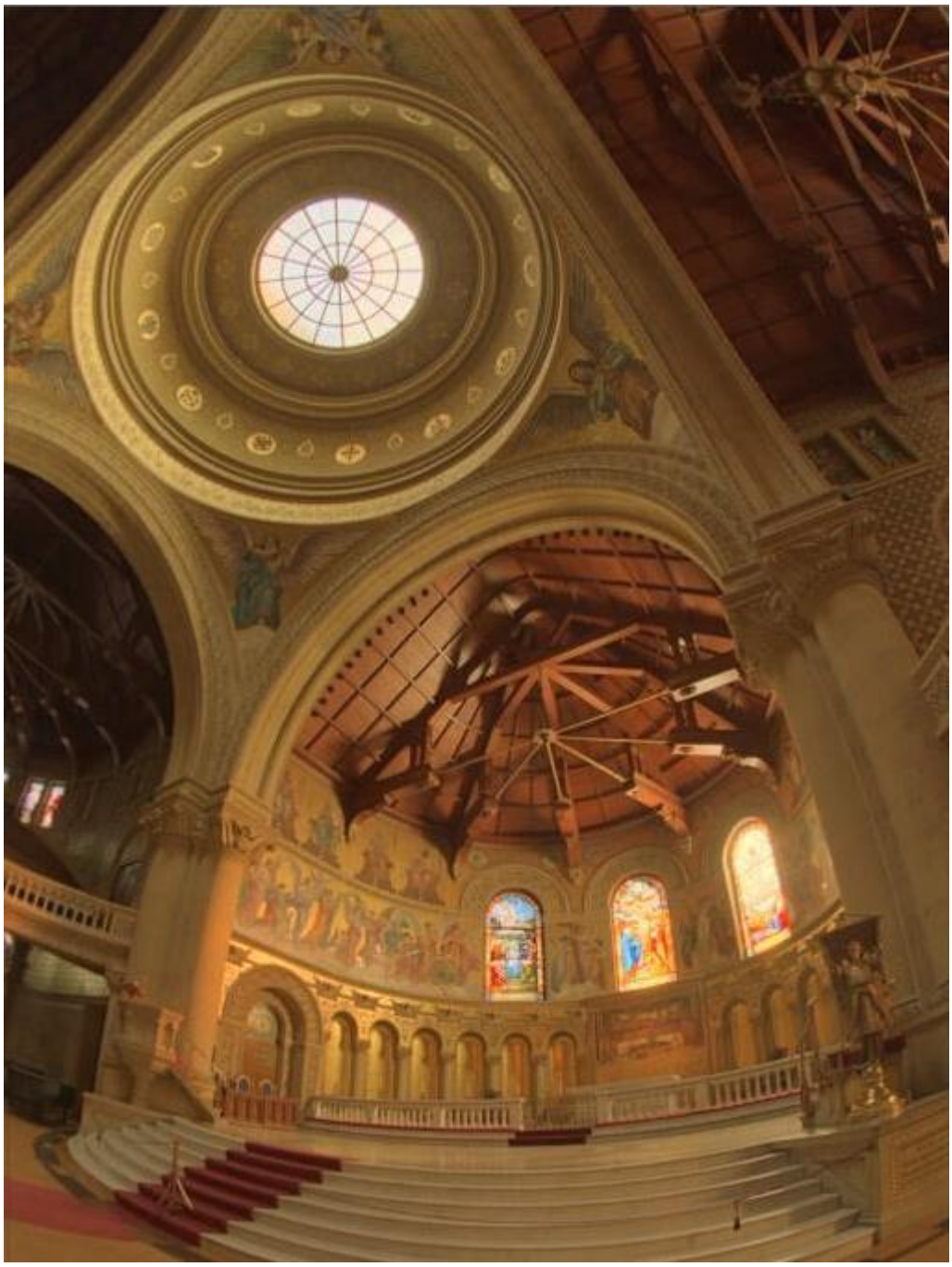
三、 实验步骤

1. 读取输入的.bmp 图像并转为 float 形式，得到线性 RGB 值;

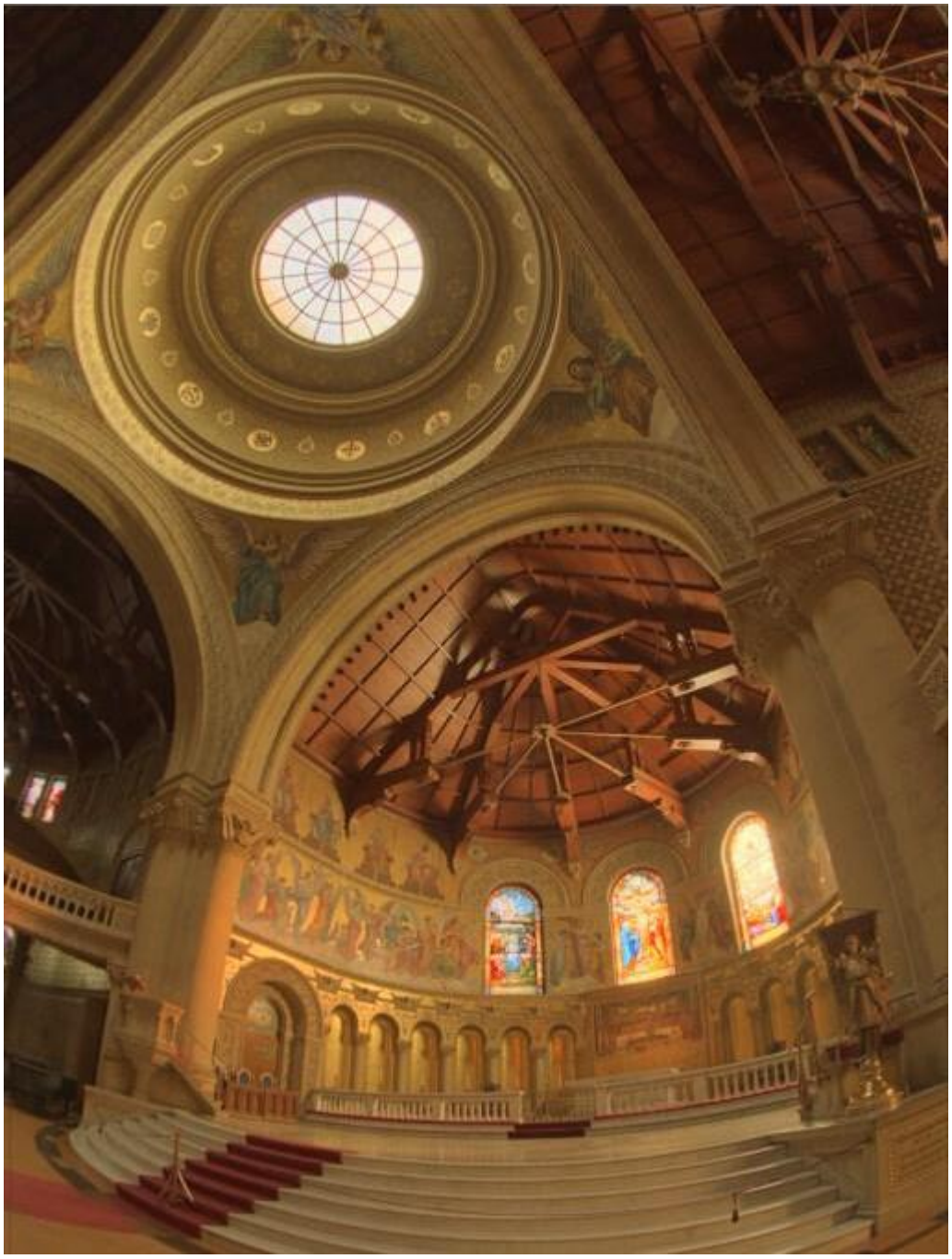
2. 将 RGB 转成 XYZ 的形式，每个像素的 Y 值就是我们要处理的亮度值,通过遍历得到 Y 的最大值。
3. 利用基于对数的 Tone mapping 的主要算法，对于图像中的每个像素点的亮度值，代入公式得到新的亮度值;
4. 把新的亮度值转化为 RGB 值，重新绘图。

四、 实验结果

原图：



Tone Mapping 后的图像：



五、 主要源程序

```
/*读取输入的.bmp 图像并转为 float 形式的线性 RGB 值：*/
//转化函数
    void rgbe2float(double *red, double *green, double *blue, unsigned char rgbe[4])
    {
            double f;
            if (rgbe[3]) {     //非零像素
               f = ldexp(1.0,rgbe[3]-(int)(128+8));
```

```
            *red = rgbe[0] * f > 0. ? rgbe[0]*f : EPSILON;
            *green = rgbe[1] * f > 0. ? rgbe[1]*f : EPSILON;
            *blue = rgbe[2] * f > 0. ? rgbe[2]*f : EPSILON;
        }
        else
            *red = *green = *blue = EPSILON;
    }
//读图片
    int RGBE_Read (FILE *fp, int *width, int *height, rgbe_header_info *info)
    {
        char buf[128];
        double tempf;
        int i;
        found_format = 0;
      if (info) {
            info->valid = 0;
            info->programtype[0] = 0;
            info->valid |= RGBE_VALID_PROGRAMTYPE;
            for(i=0;i<sizeof(info->programtype)-1;i++) {
              if ((buf[i+2] == 0) || isspace(buf[i+2]))
                    break;
              info->programtype[i] = buf[i+2];
      }
            info->programtype[i] = 0;
       }
      if (info && (sscanf(buf,"EXPOSURE=%f",&info->exposure) == 1)) {
                info->valid |= RGBE_VALID_EXPOSURE;
      }
      return RGBE_RETURN_SUCCESS;
    }
     //从缓冲器里转化为 float 形式
      for(i=0;i<scanline_width;i++) {
                rgbe[0] = scanline_buffer[i];
                rgbe[1] = scanline_buffer[i+scanline_width];
                rgbe[2] = scanline_buffer[i+2*scanline_width];
                rgbe[3] = scanline_buffer[i+3*scanline_width];
                rgbe2float(&data[RGBE_DATA_RED],&data[RGBE_DATA_GREEN],
                  &data[RGBE_DATA_BLUE],rgbe);
                data += RGBE_DATA_SIZE;
      }
      num_scanlines--;

       free(scanline_buffer);
       return RGBE_RETURN_SUCCESS;
```

```
    }
//将 RGB 转成 XYZ 的形式，每个像素的 Y 值就是我们要处理的亮度值:
    double RGB2Yxy[3][3] = { {0.5141364, 0.3238786, 0.16036376},
                             {0.265068, 0.67023428, 0.06409157},
                             {0.0241188, 0.1228178, 0.84442666}
    };
    Void rgb_Xyz (SCENE *scene, int width, int height, double *maximum, double
    *minimum, double *log_av)
    {
        int x, i;
        double W, result[3];
        double max, min;
        double sum = 0.0;
        int array_size = width * height;
        max = EPSILON;
        min = INF;
        for (x = 0; x < array_size; x++)
        {
            result[0] = result[1] = result[2] = 0.;
            for (i = 0; i < 3; i++){
                result[i] += RGB2Yxy[i][0] * scene[x].r;
                result[i] += RGB2Yxy[i][1] * scene[x].g;
                result[i] += RGB2Yxy[i][2] * scene[x].b;
            }
            if ((W = result[0] + result[1] + result[2]) > 0.) {
                scene[x].r = result[1];         // X
                scene[x].g = result[0] / W;// Y
                scene[x].b = result[1] / W;// Z
            }
            else
                scene[x].r = scene[x].g = scene[x].b = 0.;

            max = (max < scene[x].r) ? scene[x].r : max;   //图像中的亮度最大值
            min = (min > scene[x].r) ? scene[x].r : min;    //图像中的亮度最小值
              sum += log(2.3e-5 + scene[x].r); //Contrast constant Tumblin paper
        }
    }
//基于对数的 Tone mapping 的主要算法：
Void logmapping (SCENE * scene,
                 int width,
                 int height,
                 double Lum_max,
                  double Lum_min,
                 float world_lum,
```

```
                float biasParam,
                float contParam,
                float exposure,
                float white)
    {
        double Lmax, divider, av_lum, interpol, biasP, contP;
        int x, y, i, j, index;
        int nrows, ncols;
        double L;
        double exp_adapt;
        double Average;
        extern int fast;

        nrows = height;
        ncols = width;
 // Bias 参数
        exp_adapt = 1;//pow(biasParam,5);
        av_lum = exp(world_lum) / exp_adapt;
        fprintf(stderr, "world adaptation: %f \n", av_lum);

        biasP = log(biasParam)/LOG05;
        contP = 1/contParam;
        Lmax = Lum_max/av_lum;
        divider = log10(Lmax+1);

//  对每个像素的普通 tone mapping
        if (!fast) {
          for (x=0; x < nrows; x++)
            for (y = 0; y < ncols; y++) {
              index = x * ncols + y;
      //插值算法
              interpol = log (2 + bias(biasP, scene[index].r / Lmax) * 8);

             scene[index].r = ((log (scene[index].r+1)/interpol)/divider);
            }
        }
        else {
          for (x=0; x<nrows; x+=3)
            for (y=0; y<ncols; y+=3) {
              Average = 0.;
              for (i=0; i<3; i++)
                for (j=0; j<3; j++) {
                  scene[(x+i)*ncols+y+j].r /= av_lum;
                 if (exposure != 1.)
```

```
                    scene[(x+i)*ncols+y+j].r*= exposure;
                      Average += scene[(x+i)*ncols+y+j].r;
                    }
                  Average = Average / 9 - scene[x*ncols+y].r;
                  if (Average > -1 && Average < 1) {
                    interpol =
                        log(2+pow(scene[(x+1)*ncols+y+1].r/Lmax, biasP)*8);
                    for (i=0; i<3; i++)
                      for (j=0; j<3; j++) {
                        index = (x+i)*ncols+y+j;
                       if (scene[index].r < 1) {
                          L=scene[index].r*(6+scene[index].r)/(6+4*scene[index].r);
                            scene[index].r = (L/interpol) / divider;
                          }
                          else if ( scene[index].r >= 1 && scene[index].r < 2) {
                            L = scene[index].r*(6+0.7662*scene[index].r)/
                                  (5.9897+3.7658*scene[index].r);
                           scene[index].r = (L/interpol) / divider;
                         }
                          else
                        scene[index].r =
                              (log (scene[index].r + 1)/interpol)/divider;
                      }
                  }
                  else {
                    for (i=0; i<3; i++)
                      for (j=0; j<3; j++) {
                        interpol =
                              log(2+pow(scene[(x+i)*ncols+y+j].r/Lmax, biasP)*8);
                         scene[(x+i)*ncols+y+j].r =
                              (log(scene[(x+i)*ncols+y+j].r+1)/interpol)/divider;
                       }
                  }
                } //y
              } // else
    }
//转化 XYZ 回 RGB 格式
double Yxy2RGB[3][3] = { {2.5651, -1.1665, -0.3986},
                         {-1.0217, 1.9777, 0.0439},
                         {0.0753, -0.2543, 1.1892}
};
```

```
Void rgb_Yxy (SCENE *scene, int width, int height, double *maximum, double
*minimum, double *log_av)
{
   int x, i;
   double W, result[3];
   double max, min;
   double sum = 0.0;
   int array_size = width * height;
   max = EPSILON;
   min = INF;
   for (x = 0; x < array_size; x++)
   {
        result[0] = result[1] = result[2] = 0.;
         for (i = 0; i < 3; i++){
              result[i] += RGB2Yxy[i][0] * scene[x].r;
              result[i] += RGB2Yxy[i][1] * scene[x].g;
              result[i] += RGB2Yxy[i][2] * scene[x].b;
         }
        if ((W = result[0] + result[1] + result[2]) > 0.) {
                        scene[x].r = result[1];          // x
           scene[x].g = result[0] / W;    // y
           scene[x].b = result[1] / W;    // y
        }
         else
           scene[x].r = scene[x].g = scene[x].b = 0.;
        max = (max < scene[x].r) ? scene[x].r : max;          // Max Luminance in Scene
        min = (min > scene[x].r) ? scene[x].r : min;          // Min Luminance in Scene
        sum += log(2.3e-5 + scene[x].r); //Contrast constant Tumblin paper
   }
   *maximum = max;
   *minimum = min;
   *log_av = (sum / (width * height));
```