

1. Benutzen und verändern Sie die aus Computergraphik bekannten Programmdateien `textur.c`, `texture.c` und `texture.h`, um das Bild `lena.rgb` anzuzeigen. Das Bild soll das Fenster ganz ausfüllen.

Die Datei `textur.c` in `anzeigetex.c` umbenennen, in der Datei `anzeigetex.c` die zwölfte Zeile in `img=LoadRGB("lena.rgb");` umändern, `make anzeigetex` und `./anzeigetex`

Angepasste display-Methode:

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex2f(-2.0, -2.0);
    glTexCoord2f(0.0, 1.0); glVertex2f(-2.0, 2.0);
    glTexCoord2f(1.0, 1.0); glVertex2f(2.0, 2.0);
    glTexCoord2f(1.0, 0.0); glVertex2f(2.0, -2.0);

    /* glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
    glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);*/
    glEnd();
    glutSwapBuffers();
}
```

2. Verändern Sie Ihre Datei so, dass dem 2-dimensionalen Charakter von Pixel-Bildern Rechnung getragen wird!  
Benutzen Sie Befehle wie `vertex2D` oder für die Perspektive `gluOrtho2D(links,rechts,unten,oben)`.

Dazu müssen die Bildkoordinaten in der `display`-Methode nochmal angepasst werden:

```
glTexCoord2f(0.0, 0.0); glVertex2f(-1.0, -1.0);
glTexCoord2f(0.0, 1.0); glVertex2f(-1.0, 1.0);
glTexCoord2f(1.0, 1.0); glVertex2f(1.0, 1.0);
glTexCoord2f(1.0, 0.0); glVertex2f(1.0, -1.0);
```

und in der `reshape`-Methode:

```
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-1, 1, -1, 1);
    // gluPerspective(50.0, 1.0*(GLfloat)w/(GLfloat)h, 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // glTranslatef(0.0, 0.0, -3.6);
}
```

3. Es gibt neben dem Anbringen von Texturen in OpenGL auch noch die Möglichkeit, Pixel-Bilder direkt anzuzeigen.  
Löschen Sie alles, was mit Textur zu tun hat. Stattdessen verwenden

Sie:

in init:

```
img = LoadRGB("...");  
glPixelStorei(GL_UNPACK_ALIGNMENT,1);
```

dadurch wird die Speicherart festgelegt

in display:

```
glRasterPos2i(0,0);
```

positioniert die linke untere Ecke in die Koordinate (0,0)

```
glDrawPixels(img->width,img->height,GL_RGB,GL_UNSIGNED_BYTE,img->imgData);
```

zeigt das Bild an.

```
void myinit(void)  
{  
    glClearColor (0.0, 0.0, 0.0, 0.0);  
    glEnable(GL_DEPTH_TEST);  
    glDepthFunc(GL_LESS);  
  
    img=LoadRGB("lena.rgb");  
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);  
}  
  
void display(void)  
{  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
    glRasterPos2i(0,0);  
  
    glDrawPixels(img->width,img->height,GL_RGB,GL_UNSIGNED_BYTE,img->imgData);  
  
    /* glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);  
    glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);  
    glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);  
    glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);*/  
    glutSwapBuffers();  
}
```

## B.3 Rasterung und Quantisierung

Wir wollen Rasterung und Quantisierung eines Bildes in OpenGL imitieren. Achten Sie möglichst auf Wiederverwendbarkeit Ihrer Funktionen!

1. Speichern Sie ein OpenGL-Bild in ein zweidimensionales Array, dessen Zeilen den Zeilen des Bildes (x, Breite) und dessen Spalten den Spalten des Bildes (y, Höhe) entspricht!



```

        break;
    case '1' : if (rastern) {raster(2);}
               if (quantisieren) {quant(2);}
               break;
    case '2' : if (rastern) {raster(4);}
               if (quantisieren) {quant(4);}
               break;
    case '3' : if (rastern) {raster(8);}
               if (quantisieren) {quant(8);}
               break;
    case '4' : if (rastern) {raster(16);}
               if (quantisieren) {quant(16);}
               break;
    case '5' : if (rastern) {raster(32);}
               if (quantisieren) {quant(32);}
               break;
    case '6' : if (rastern) {raster(64);}
               if (quantisieren) {quant(64);}
               break;
    case '7' : if (rastern) {raster(128);}
               if (quantisieren) {quant(128);}
               break;
    case 'e' : RGBImage3D();
               raster(1);
               quant(7);
               break;
}
glutPostRedisplay();
}

```

#### 4. Rasterung:

Schreiben Sie eine Funktion, die  $l \times l$  Pixel zusammenfasst.  $l$  soll eingegeben werden können (Einfacherweise nur Zweierpotenzen  $< 256$ ). Dazu wird innerhalb des Arrays jeweils über Quadrate von  $l \times l$  Pixel gemittelt und allen diesen Pixeln im Quadrat dieser Mittelwert zugewiesen.

```

void raster(int wert)
{
    int x, y, matrix_x, matrix_y, k, summe;

    for (x = 0; x < img->width; x=x+wert)
    {
        for (y = 0; y < img->height; y=y+wert)
        {
            summe = 0;
            for (matrix_x = 0; matrix_x < wert; matrix_x++)
            {
                for (matrix_y = 0; matrix_y < wert; matrix_y++)
                {
                    summe += inhalt[x+matrix_x][y+matrix_y];
                }
            }
        }
    }
}

```

```

    }
    summe = summe / (wert * wert);
    for (matrix_x = 0; matrix_x < wert; matrix_x++)
    {
        for (matrix_y = 0; matrix_y < wert; matrix_y++)
        {
            for (k = 0; k < 3; k++)
            {
                inhalt3D[x+matrix_x][y+matrix_y][k] = summe;
            }
        }
    }
}

```

und Anpassung in der Display-Methode:

```

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glRasterPos2i(0,0);

    glDrawPixels(img->width,img->height,GL_RGB,GL_UNSIGNED_BYTE,inhalt3D);

    /* glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
    glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);*/
    glutSwapBuffers();
}

```

##### 5. Quantisierung:

Es soll die Anzahl der Graustufen (Zweierpotenzen < 256) vorgegeben werden. Dann ergibt sich für jeden Pixel mit

$factor = 256 / \text{graustufen}$

$hilf = a[i][j] / factor$

$a[i][j] = hilf * factor$

ein neuer Grauwert. Die neuen Grauwerte liegen in unserem alten Grauwertsystem  $\{0, \dots, 255\}$  um  $factor$  auseinander. Dadurch wird eine Grauwertmenge von  $graustufen$  Grauwerten simuliert.

```

void quant(int wert)
{
    int x, y, k, hilf;

    wert = 256 / wert;

    for (x=0; x < img->width; x++)
    {
        for (y=0; y < img->height; y++)
        {

```

```
for (k=0; k < 3; k++)  
{  
    hilf = inhalt[x][y] / wert;  
    inhalt3D[x][y][k] = hilf * wert;  
}  
}  
}
```