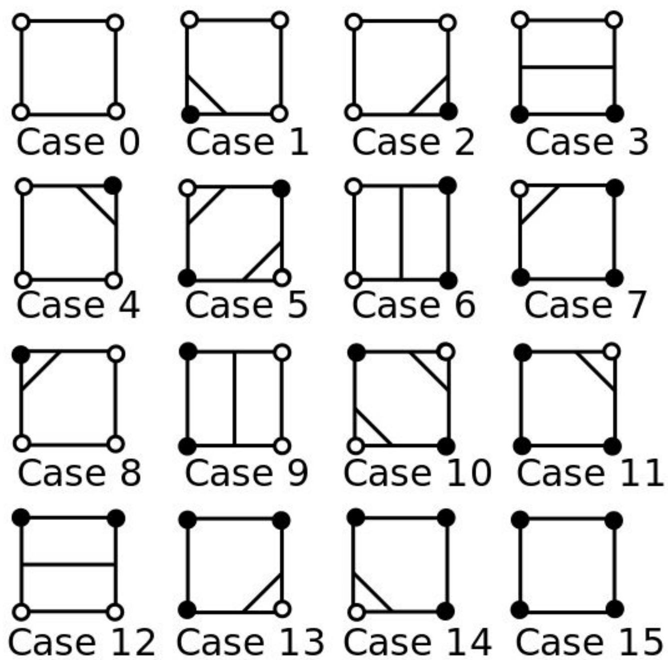
Assignments 2+3

174455 - Konstantinos Karatsenidis

General remarks:

- Changed LoadData to keep the min and max value
- Changed DrawContour signature to keep aspect ratio
- Global contour stores 2d point tuples for marching squares contour lines
- Global vertices stores 3d points of triangles
- Global normals stores normal values of the vertices

Marching Squares



There are 7 cases that need lines drawn due to symmetry. Only case 5 (and 10) need 2 lines to be drawn. MarchingSquares selects the appropriate surrounding values and calls GetLinePoints which decides which is the current square case, and calculates the 2d points of the contour. The position of each point on the square's edge is determined by the weights of the corner values.

Marching Cubes

With a similar concept, but in 3d space, MarchingCubes for each cube, created by adjacent points in space, collects the vertex and normal values of its edges. Then it finds the appropriate cube case and according to the edgeTable3D calculates the triangle points for the vertex and the normal values. The values are normalized between [-1, 1].

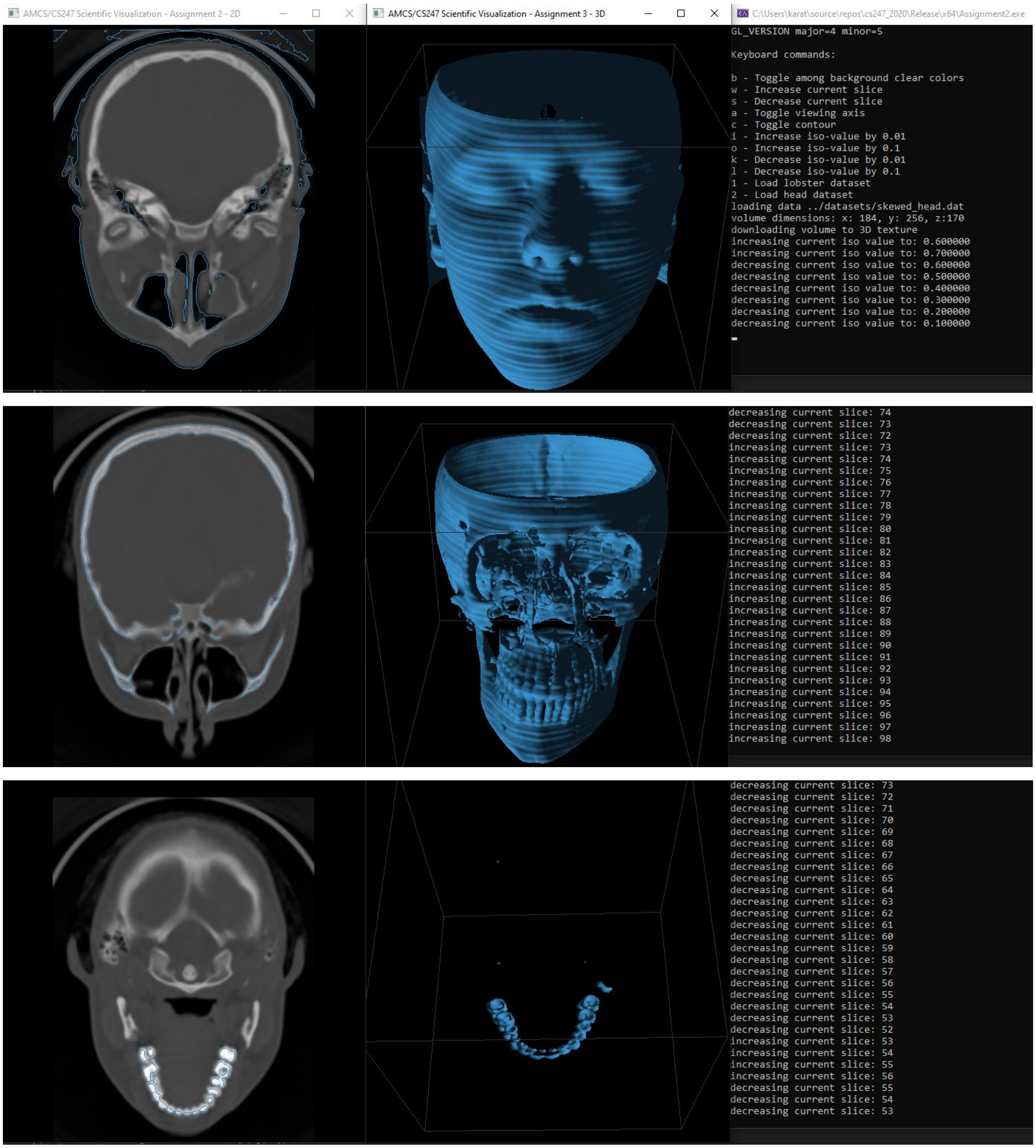


Table offset provides a simple way to access the neighbors of a point in an iterative way.
For each point, collect neighboring points coordinates p and values val.
Using these values, determine cube case cubeIndex.

```
for (int x = 0; x < vol_dim[0] - 1; x++) {
    for (int y = 0; y < vol_dim[1] - 1; y++) {
        for (int z = 0; z < vol_dim[2] - 1; z++) {
            Vertex p[8];
            double val[8];
            uint8_t cubeIndex = 0;
            for (uint8_t i = 0; i < 8; i++) {
                p[i].x = x + offset[i].x;
                p[i].y = y + offset[i].y;
                p[i].z = z + offset[i].z;
                val[i] = data(p[i].x, p[i].y, p[i].z);

                if (val[i] < isoValue) {
                    cubeIndex |= 1u << i;
                }
            }
```

For each edge of the cubes, determine if a line should be drawn and calculate its points.

```
            Vertex vertlist[12];
            Vertex normaList[12];
            for (uint8_t i = 0; i < 12; i++) {
                uint16_t bit = 1u << i;
                if (edgeTable3D[cubeIndex] & bit) {
                    uint8_t a = points[i][0];
                    uint8_t b = points[i][1];

                    Vertex& p1 = p[a];
                    Vertex& p2 = p[b];
                    double v1 = val[a];
                    double v2 = val[b];
                    Vertex n1 = midpointDiff(p1.x, p1.y, p1.z);
                    Vertex n2 = midpointDiff(p2.x, p2.y, p2.z);
                    float mu = (isoValue - v1) / (v2 - v1);

                    Vertex& vertex = vertlist[i];
                    vertex.x = p1.x + mu * (p2.x - p1.x);
                    vertex.y = p1.y + mu * (p2.y - p1.y);
                    vertex.z = p1.z + mu * (p2.z - p1.z);

                    Vertex& normal = normaList[i];
                    normal.x = n1.x + mu * (n2.x - n1.x);
                    normal.y = n1.y + mu * (n2.y - n1.y);
                    normal.z = n1.z + mu * (n2.z - n1.z);
                }
```

Table points is used to easily find the edges of each case.
For both methods, iso-value is normalized from [min, max] to [0, 1]. That means there will always be something to draw. For low values, we can see skin tissue, medium values reveal the

skull, and the larger values show only the teeth. The contour line is displayed only on the slice where the object lies.