5110597: Computaional Frabication

Final Project: Lithophane

Member:

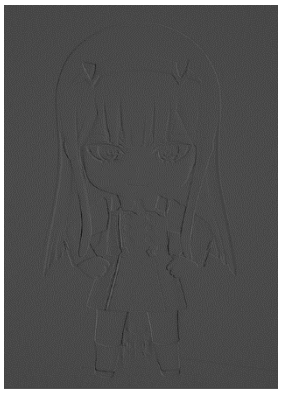
5931040421 Mr.Pongsathorn Kerdphol

Description:

โปรเจคนี้จะเป็นการสร้าง 3D model จากรูปภาพ เมื่อปริ้นออกมาแล้วด้วย 3D Printer จะลักกษณะเหมือนงานแกะสลัก เมื่อนำผลงานมาวางหน้าแหล่งกำเนิดแสง จะเกิดเป็นรูปภาพ

รูปตัวอย่าง

รูปขาวดำ



รูป 3D model (Top view)

Code:

main.py

```
    v1 = []
    v2 = []
    # top
    for i in range(height-1):
        v1 = []
        v2 = []
        for j in range(width):
            v1.append((j * output_width / width,
                    i * output_height / height,
                    (255 - img[i, j]) * layer / 255 + thickness))
            v2.append((j * output_width / width,
                    (i+1) * output_height / height,
                    (255 - img[i+1, j]) * layer / 255 + thickness))
        geometry.quad(v1, v2)

    # side width
    v1 = []
    v2 = []
    for i in range(width):
      v1.append((i * output_width / width,
        0 * output_height / height,
        (255 - img[0, i]) * layer / 255 + thickness))
      v2.append((i * output_width / width,
        0 * output_height / height,
        0))
    geometry.quad(v1, v2)

    v1 = []
    v2 = []
    for i in range(width):
      v1.append((i * output_width / width,
        (height-1) * output_height / height,
        (255 - img[height-1, i]) * layer / 255 + thickness))
      v2.append((i * output_width / width,
        (height-1) * output_height / height,
        0))
    geometry.quad(v1, v2)

    # side height
    v1 = []
    v2 = []
    for i in range(height):
      v1.append((0 * output_width / width,
        i * output_height / height,
        (255 - img[i, 0]) * layer / 255 + thickness))
      v2.append((0 * output_width / width,
        i * output_height / height,
        0))
    geometry.quad(v1, v2)

    v1 = []
    v2 = []
    for i in range(height):
      v1.append(((width-1) * output_width / width,
        i * output_height / height,
        (255 - img[i, width-1]) * layer / 255 + thickness))
      v2.append(((width-1) * output_width / width,
        i * output_height / height,
        0))
    geometry.quad(v1, v2)

    # bottom
    v1 = []
    v2 = []
    v1.append((0 * output_width / width, 0 * output_height / height, 0))
    v2.append((0 * output_width / width, (height-1) * output_height / height, 0))
    v1.append(((width-1) * output_width / width, 0 * output_height / height, 0))
    v2.append(((width-1) * output_width / width, (height-1) * output_height / height, 0))
    geometry.quad(v1, v2)

    return geometry
```

geo.py

```
                (b, c, a) if b < a and c < a and b <= c else (c, b, a)
            )
        )
    )

class GEO:
    """docstring for GEO"""
    def __init__(self):
        self.points = {}
        self.edges = set()
        self.faces = set()

    def _add_points(self, point = (0, 0, 0)):
        if point not in self.points.keys():
            self.points[point] = len(self.points)
        return self.points[point]

    def quad(self, v1, v2):
        if len(v1) != len(v2): return
        if len(v1) == 1:
            p1 = self._add_points(v1[0])
            p2 = self._add_points(v2[0])
            self.edges.add(sorted_tuple(v1[0], v2[0]))
        else:
            for i in range(1, len(v1)):
                i0 = self._add_points(v1[i-1])
                i1 = self._add_points(v2[i-1])
                i2 = self._add_points(v1[i])
                i3 = self._add_points(v2[i])

                # self.edges.add(sorted_tuple(i0, i1))
                # self.edges.add(sorted_tuple(i0, i2))
                # self.edges.add(sorted_tuple(i0, i3))
                # self.edges.add(sorted_tuple(i1, i2))
                # self.edges.add(sorted_tuple(i1, i3))
                # self.edges.add(sorted_tuple(i2, i3))

                self.faces.add(sorted_tuple_3(i0, i1, i2))
                # self.faces.add(sorted_tuple_3(i0, i1, i3))
                # self.faces.add(sorted_tuple_3(i0, i2, i3))
                self.faces.add(sorted_tuple_3(i1, i2, i3))

    def save_stl(self, filename = 'geo.stl'):
        # Create the mesh
        faces = np.array(list(self.faces))
        output = mesh.Mesh(np.zeros(faces.shape[0], dtype=mesh.Mesh.dtype))
        points = {}
        for k in self.points.keys():
            t = self.points[k]
            points[t] = np.array(list(k))

        for i, f in enumerate(faces):
            for j in range(3):
                output.vectors[i][j] = points[f[j]]

        # Write the mesh to file "cube.stl"
        output.save(filename)


    def __str__(self):
        return "points size: {}, edges size: {}, faces size: {}".format(len(self.points), len(self.edges), len(self.faces))
```

ในไฟล์ Main.py จะเกี่ยวกับการคำนวณจุด point cloud ของแต่ละ pixel เพื่อไปสร้าง mesh

ในไฟล์ Geo.py จะเกี่ยวกับการสร้าง mesh และ stl file