Teddy: A Sketching Interface for 3D Freeform Design

資科三 莊彩彥 資科四 郭沛澐

參考論文

題目 : Teddy: A Sketching Interface for 3D Freeform Design

作者 : Takeo Igarashi , Satoshi Matsuoka , Hidehiko Tanaka

出處 : SIGGRAPH 1999

論文介紹

摘要

本篇論文重點在實作如何僅利用平面繪畫的線條，計算出立體的形狀，讓使用者能夠快速產生3D的模型。

輸入資料類型

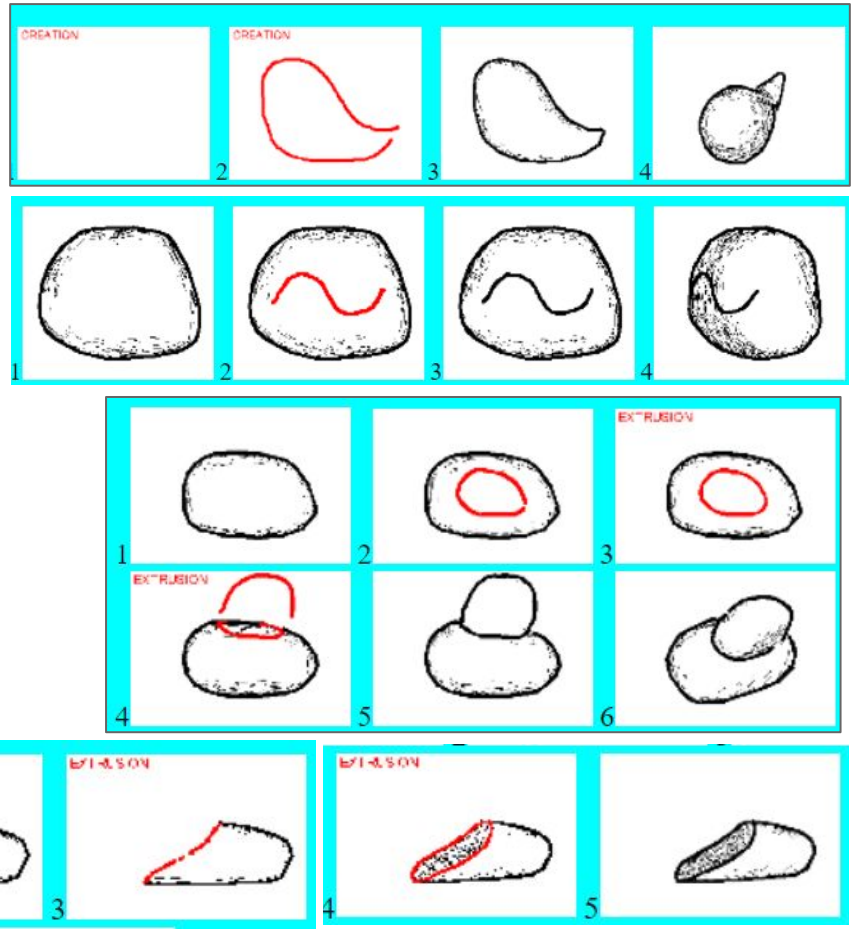
平面畫面中所繪製的線條

架構

依據使用者的動作依據使用者的操作，主要可以分成以下五個部分：

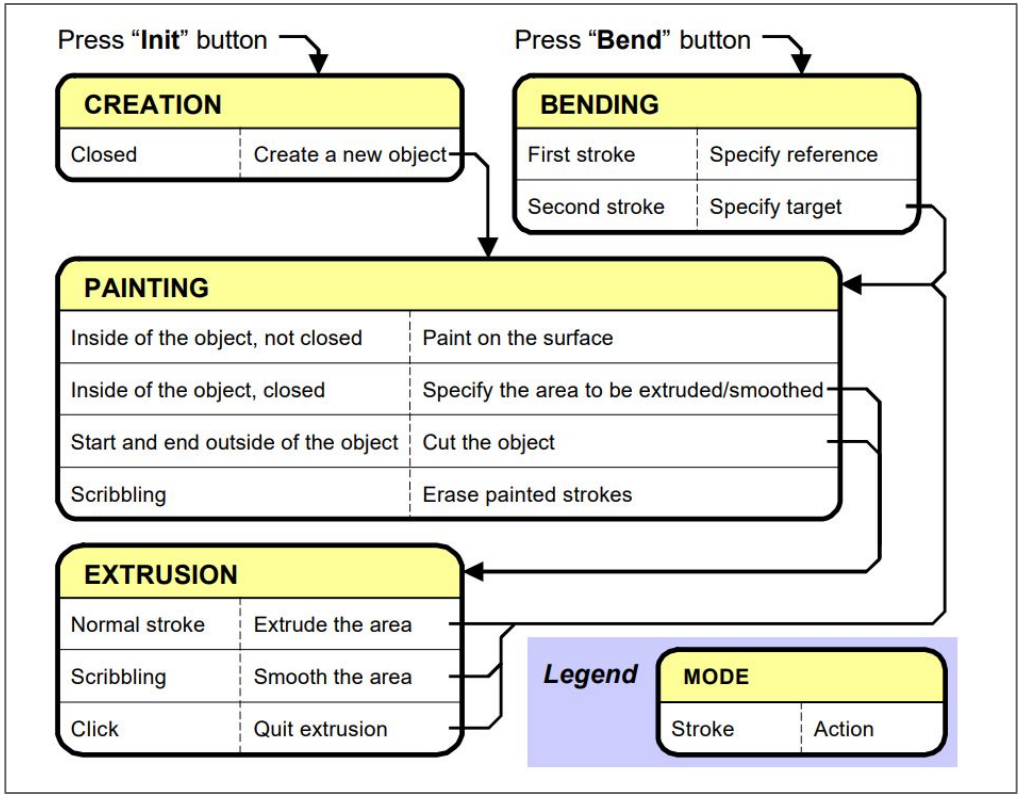
1. Creation: 使用者的第一筆畫
2. Painting: 在物體上新增線段
3. Extrusion: 新增物體上突出的區塊
4. Cutting: 切除不要的部分
5. Bending: 使物體彎折

右圖從上到下分別展示了1至4的操作



論文實作方法

使用者介面架構

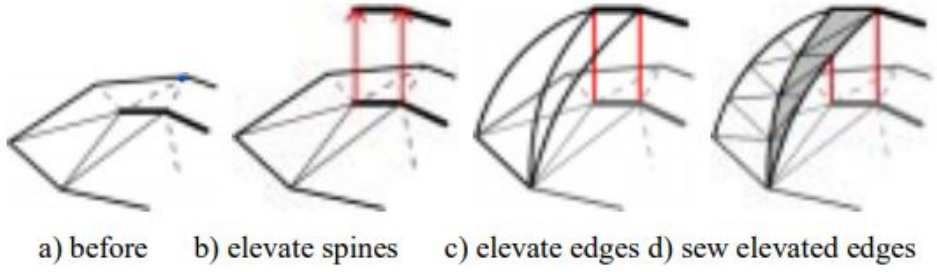

Press “Init” button
Press “Bend” button
CREATION
Closed
Create a new object
BENDING
First stroke
Specify reference
Second stroke
Specify target
PAINTING
Inside of the object, not closed
Paint on the surface
Inside of the object, closed
Specify the area to be extruded/smoothed
Start and end outside of the object
Cut the object
Scribbling
Erase painted strokes
EXTRUSION
Normal stroke
Extrude the area
Scribbling
Smooth the area
Click
Quit extrusion
Legend
MODE
Stroke
Action

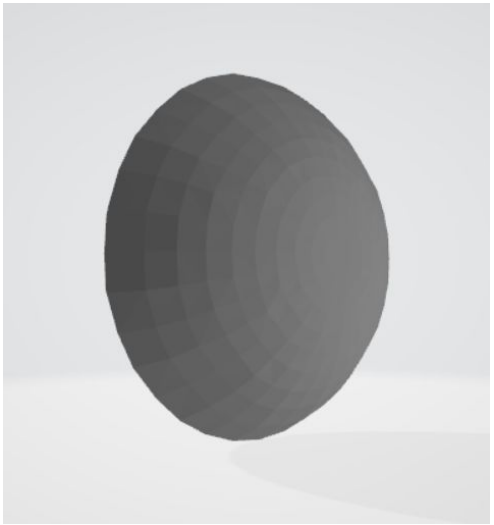
程式實作流程

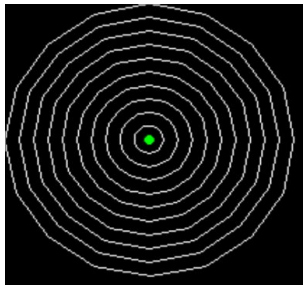
Create a New Object

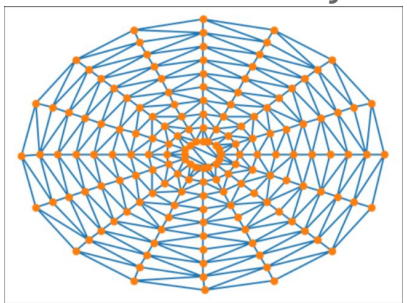
使用者畫出封閉形狀

- ➔ 紀錄形狀的頂點
- ➔ 計算chordal axis
- ➔ 抬升主軸
- ➔ 沿著主軸按比例計算縮小的形狀
- ➔ 按比例抬升 / 計算constrained Delaunay triangulation









程式實作流程

Painting

將使用者的筆劃分段

- ➔ 計算線段的左右端點與相機位置所形成的向量和物體的交點
- ➔ 若兩點連線與物體表面的線段有交點
  利用螢幕中的交點與相機位置形成的向量
  計算與物體表面線段的交點座標
- ➔ 將所有點連接

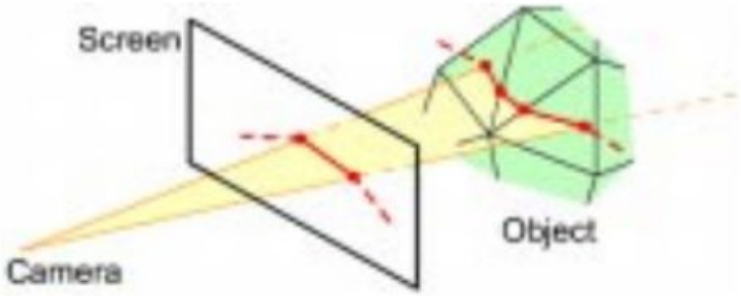


Figure 16: Construction of surface lines.

程式實作流程

Extrusion

使用者畫出base ring和延伸範圍

➔ 利用**Painting**中的方法找到base ring在原物件上的投影面
- ◆ 投影面通過base ring的重心，並且盡可能平行於畫面

➔ 根據extruding stroke決定base ring延伸方向

➔ 依據extruding stroke調整base ring的大小

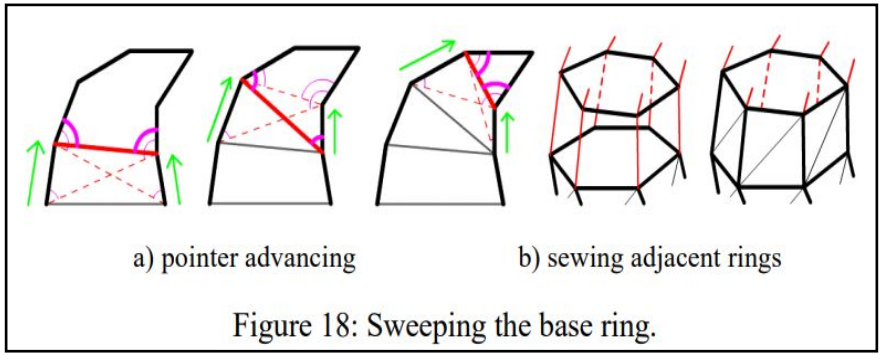


Figure 18: Sweeping the base ring.

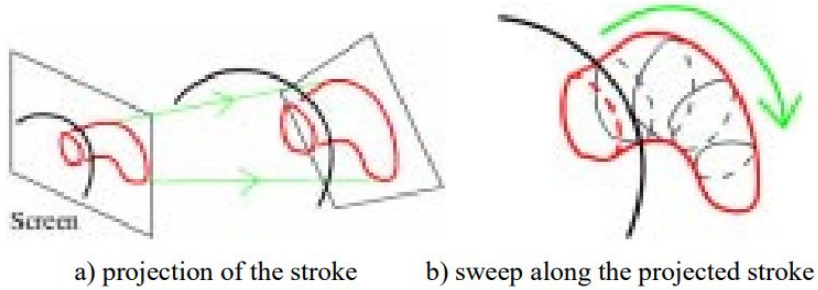


Figure 17: Extrusion algorithm.

程式實作流程

Cutting

將使用者的筆劃根據轉折點分段

- ➔ 計算線段的左右端點與相機位置所形成的向量和物體的交點
- ➔ 若兩點連線與物體表面的線段有交點 (包含看不見的那側)
  利用螢幕中的交點與相機位置所形成的向量，計算與物體表面線段的交點座標
- ➔ 將所有點連接，重新計算被切割的平面的CDT
- ➔ 刪除切割筆畫的右半邊的部分

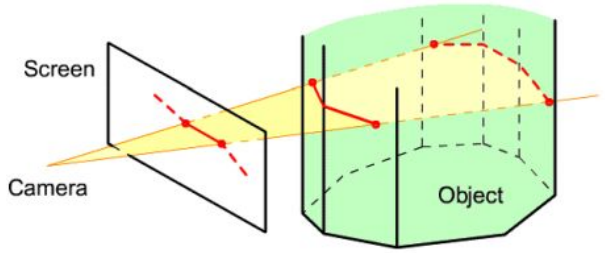


Figure 20: Cutting.

程式實作流程

Smoothing

使用者圈出範圍

➔ 把該範圍投影到XY平面上
➔ 計算投影面的CDT
➔ 將平面上的點延Z軸進行抬升
➔ 重新生成一個平滑表面

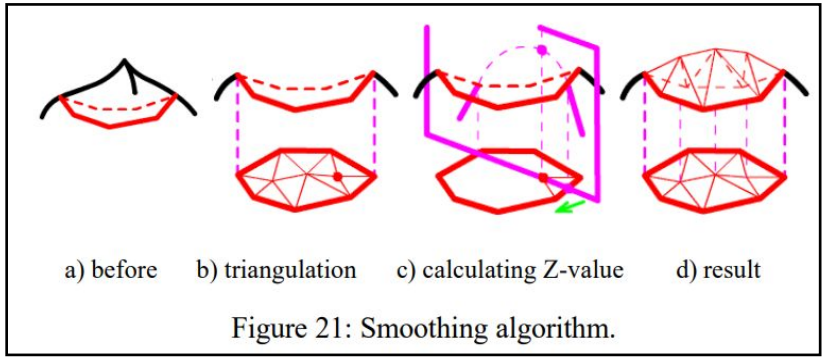
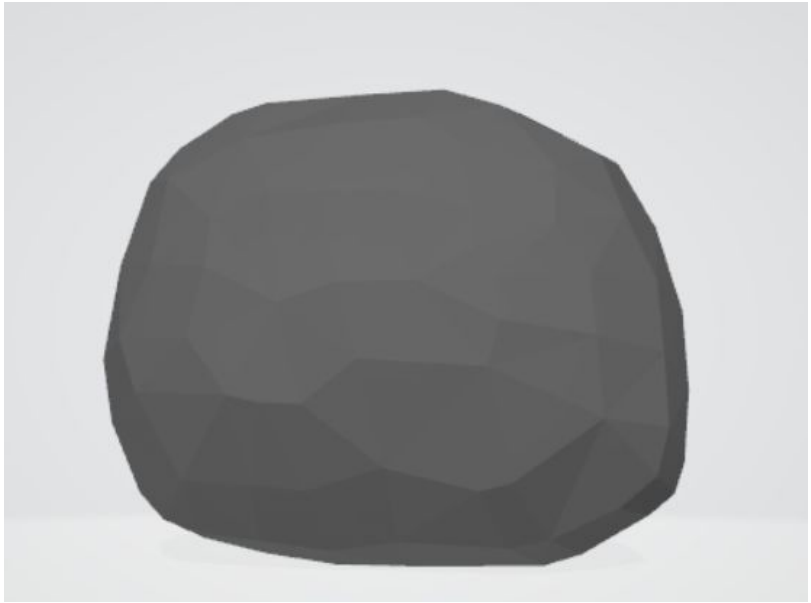


Figure 21: Smoothing algorithm.

提案預期成果

預期實作成果



預期改進的方向

使用介面

1. 增加初始化使用者視角的功能
2. 遇到無法計算的情況要報錯並取消該次操作

實際實作內容

實作內容

- 使用者介面
- 實作論文 Creation
- 實作論文 Painting

我們的程式：https://github.com/Tony891017/1112_CG_Final

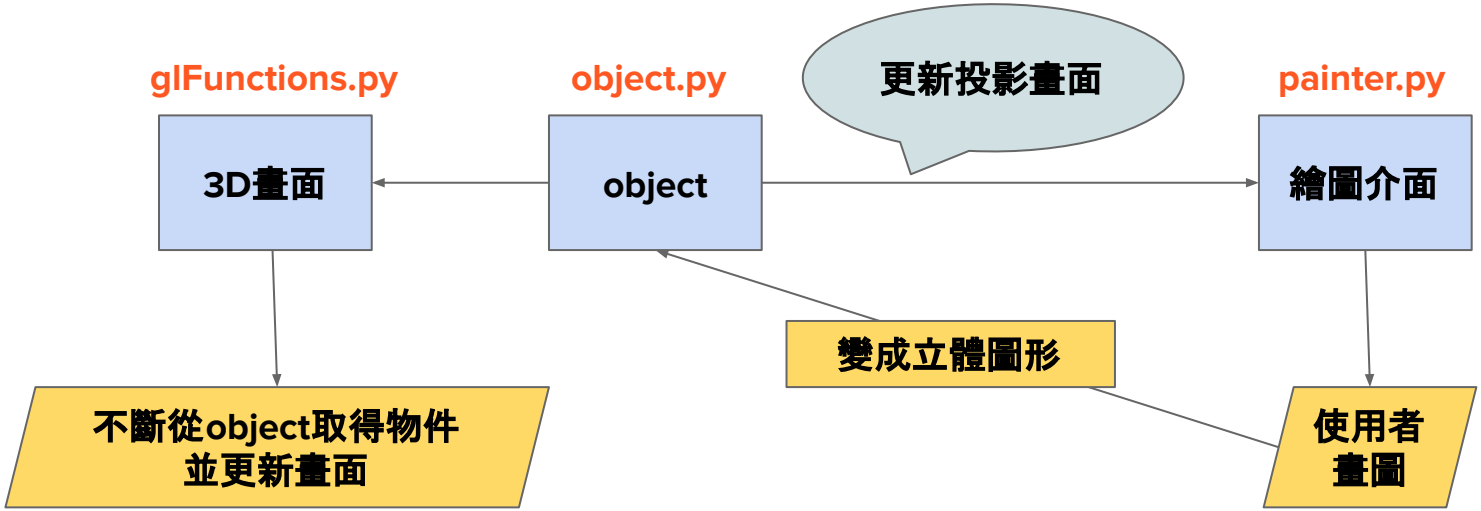
（下載後執行 python object.py）

實作內容—
使用者介面

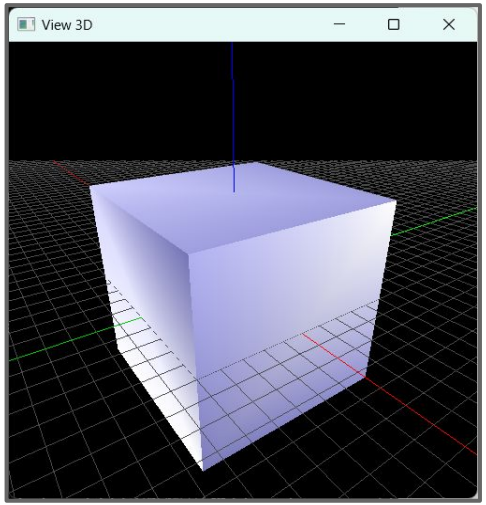
架構

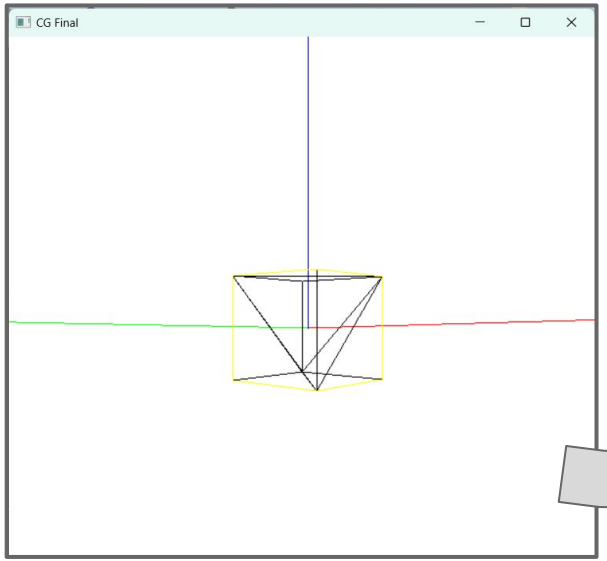
使用工具：pyqt6、pyopengl

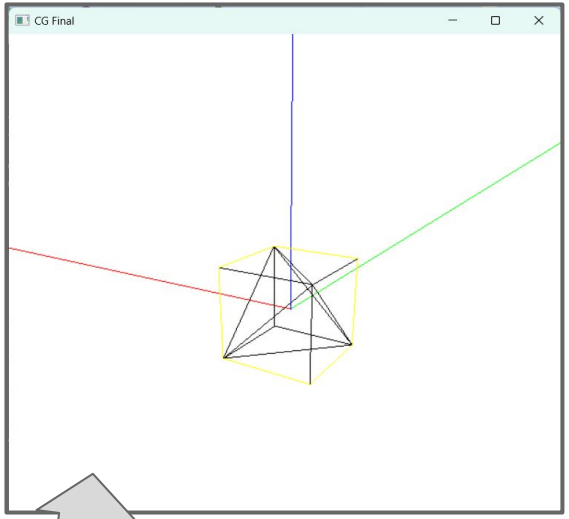
三個class與其對應的程式檔案：



介面







可做視角的轉換

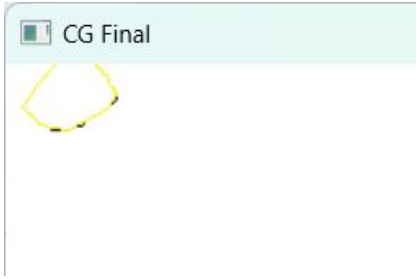
3D畫面

繪圖介面

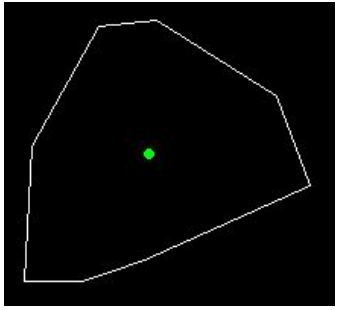
使用者可利用w、a、s、d在3D視角中移動x、y座標；以及使用i、j、k、l、在兩種介面中調整視角

實作内容—
Creation

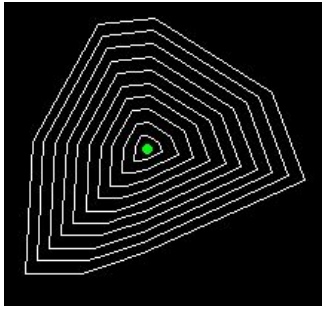
紀錄形狀頂點



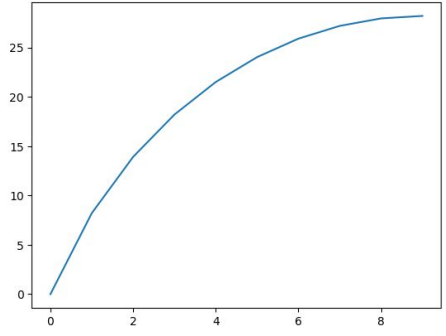
計算重心



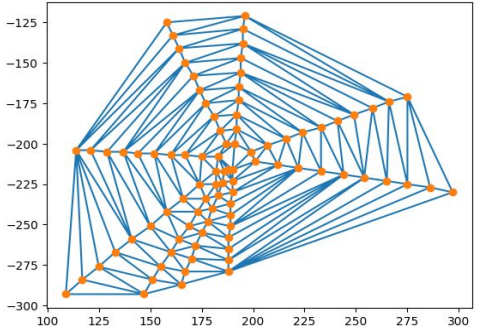
沿著主軸按比例縮小形狀



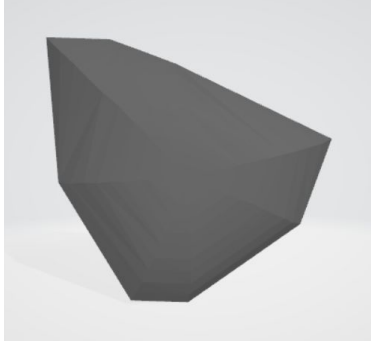
按與重心距離抬升高度



計算CDT



建立立體物件



紀錄形狀及頂點

使用者畫圖

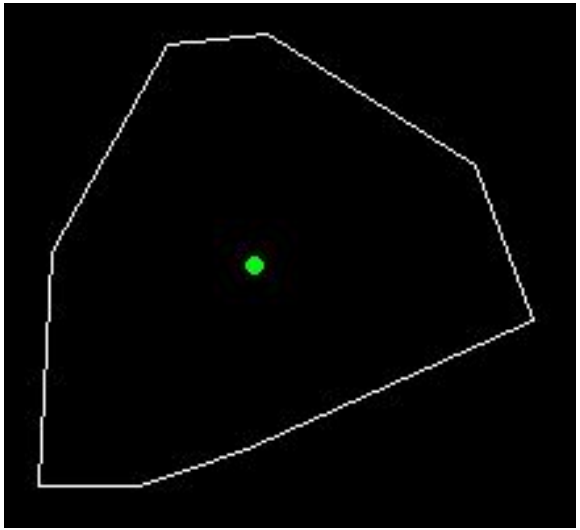

```python
    # MOUSE EVENT
    def mousePressEvent(self, event):
        self.setPenColor('#ffee00')
        self.__history_canvas.append(self.__label.pixmap())
```

```python
    def mouseMoveEvent(self, event): # 滑鼠按下後才啟動
        mx = int(QEnterEvent.position(event).x())
        my = int(QEnterEvent.position(event).y())
        if self.__path:
            if [mx, my] == self.__path[-1]: return
            self.painting([self.__path[-1], [mx, my]])
        self.__path.append([mx, my])
```

計算重心

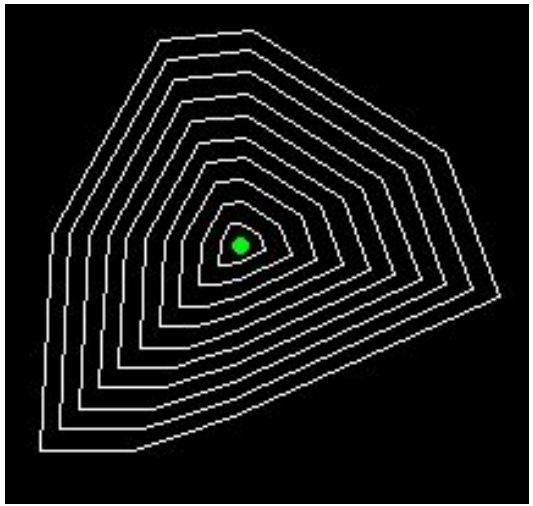
```
#重心
gx, gy = centerOfGravity(points)
```

```
def centerOfGravity(vertices):
  m = cv2.moments(vertices)
  mx = int(m['m10'] / m['m00'])
  my = int(m['m01'] / m['m00'])
  return mx, my
```



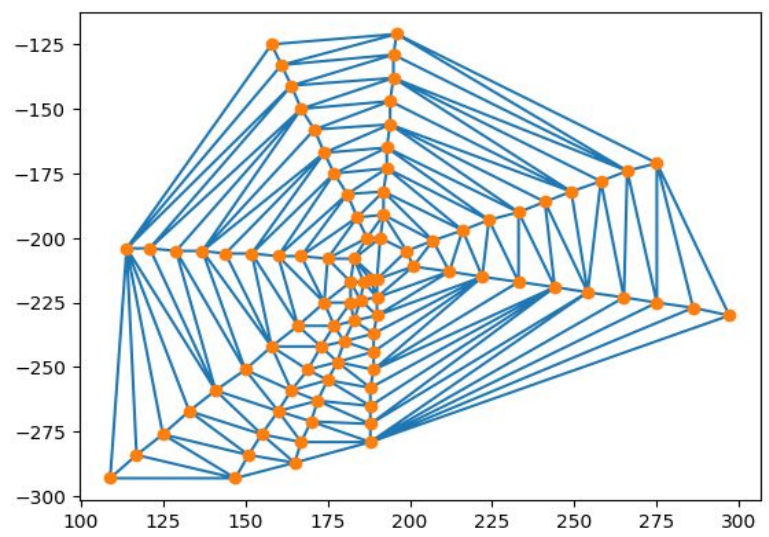
抬升主軸 & 沿著主軸按比例縮小形狀

```python
# 等高線
def elevation(vertices, segments=10, a=1, b=100): #b決定高度(float)
  gx, gy = centerOfGravity(vertices)
  center = np.array([gx, gy])

  a2 = a * a
  b2 = b * b
  fx = lambda x: math.sqrt((1 - x*x/a2) * b2)

  height = np.array([fx(a - a/segments)])
  contours = np.array([vertices])

  for i in range(1, segments):
    height = np.append(height, [fx(a - a*(i + 1)/segments)], axis=0)
    scaled_points = scaling(center, vertices, 1 - i/segments)
    contours = np.append(contours, [scaled_points], axis=0)
  height -= fx(a - a/segments)

  height = np.repeat(height, vertices.shape[0], axis=0)
  height = np.reshape(height, (segments, vertices.shape[0], 1))
  elevated_vertices = np.append(contours, height, axis=2)

  return elevated_vertices
```



計算CDT

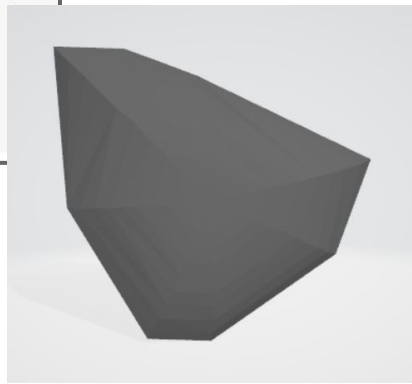
```
segs, nps, dim = contour_lines.shape
elevated_points = np.reshape(contour_lines, (segs * nps, dim))

tri = spatial.Delaunay(elevated_points[:, 0:2])

xs = elevated_points[:,0]
ys = -elevated_points[:,1]
```



寫入obj

```
#OBJ file output
output_file = open("final_test1.obj", "w")
for v in elevated_points:
    output_str = "v"
    for x in v:
        output_str += " " + str(x)
    output_str += "\n"
    output_file.write(output_str)

mirror_points = elevated_points
mirror_points[:, 2] *= -1
for v in mirror_points:
    output_str = "v"
    for x in v:
        output_str += " " + str(x)
    output_str += "\n"
    output_file.write(output_str)
```

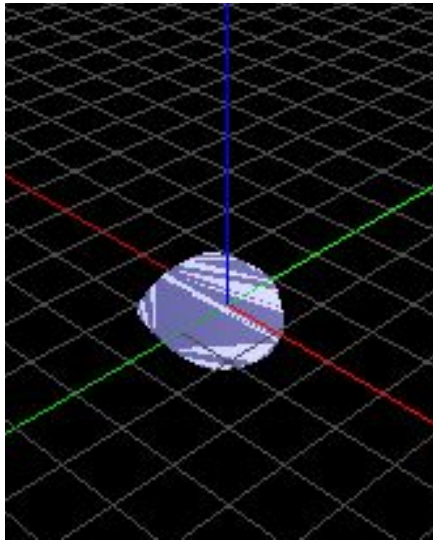
```
for i in tri.simplices:
    output_str = "f"
    for j in i:
        output_str += " " + str(j + 1)
    output_str += "\n"
    output_file.write(output_str)

offset = len(elevated_points)
for i in  tri.simplices:
    output_str = "f"
    for j in i:
        output_str += " " + str(j + 1 + offset)
    output_str += "\n"
    output_file.write(output_str)

output_file.close()
```



用OpenGL畫圖

```
if self.obj is not None:
    points, tri_indices = self.obj.getObj()
    for tri in tri_indices:
        vertices = [points[tri[0]], points[tri[1]], points[tri[2]]]
        self.drawTriangle(vertices, color)
```

```
def drawTriangle(self, vertices, color):
    glBegin(GL_TRIANGLES)
    #glColor3f(color[0], color[1], color[2])
    glMaterialfv(GL_FRONT, GL_AMBIENT, [color[0], color[1], color[2], 1.0])
    glMaterialfv(GL_FRONT, GL_DIFFUSE, [1.0, 1.0, 1.0, 1.0])
    glMaterialfv(GL_FRONT, GL_SPECULAR, [0.1, 0.1, 0.1, 1.0])
    for v in vertices:
        glVertex3f(v[0], v[1], v[2])
    glEnd()
```



實作內容—
Painting

定義直線

```
line_points = [
    (0, 1),  # (x0, y0)
    (2, 1)   # (x1, y1)
]
```

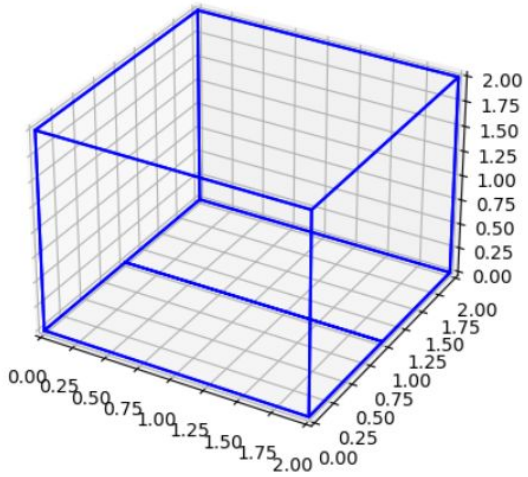
選定要投影的面

```
for point in line_points:
    x = point[0]
    y = point[1]
    z = 0  # 投影到平面上的z坐標
```

處理超過物體範圍

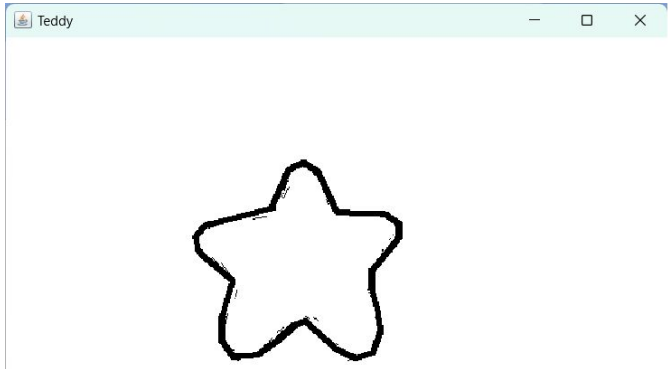
```
# 把超出範圍的去掉
if x > max:
    x = max
elif x < min:
    x = min

if y > max:
    y = max
elif y < min:
    y = min
```

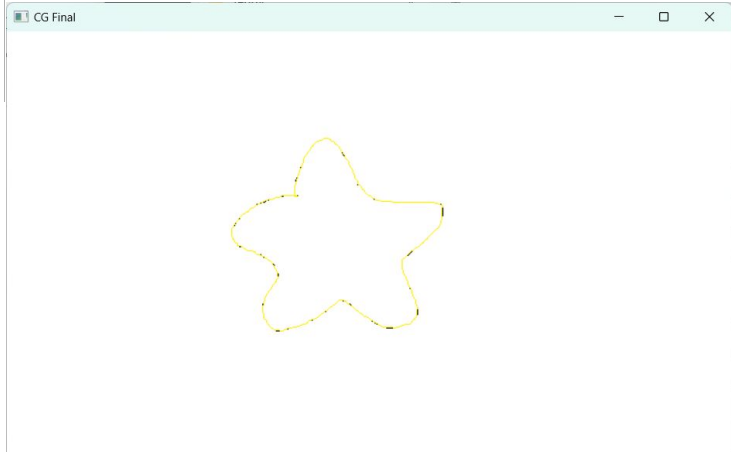
三維的端點加入obj

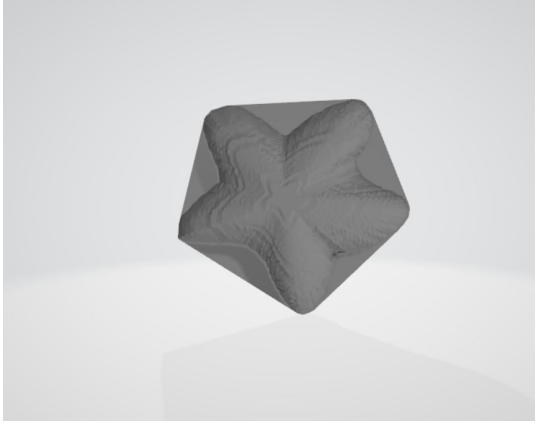


結果比較

初始筆畫生成形狀比較

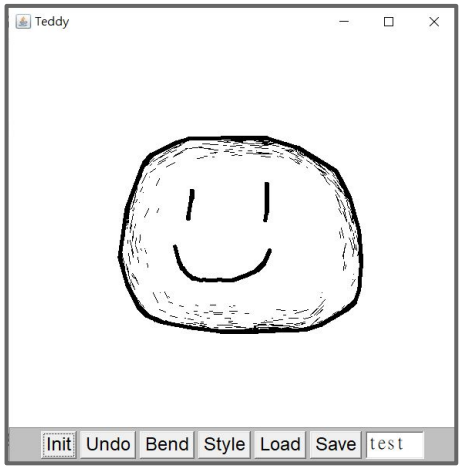




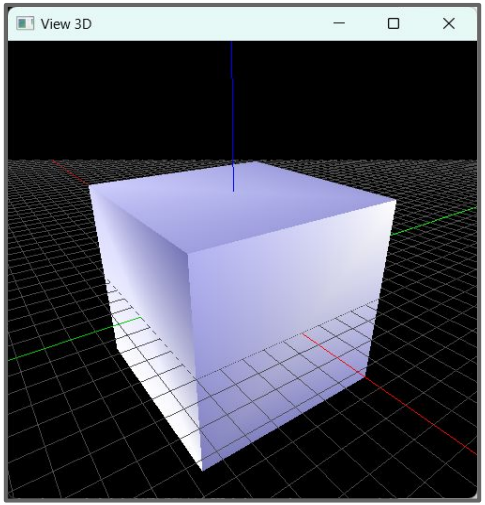


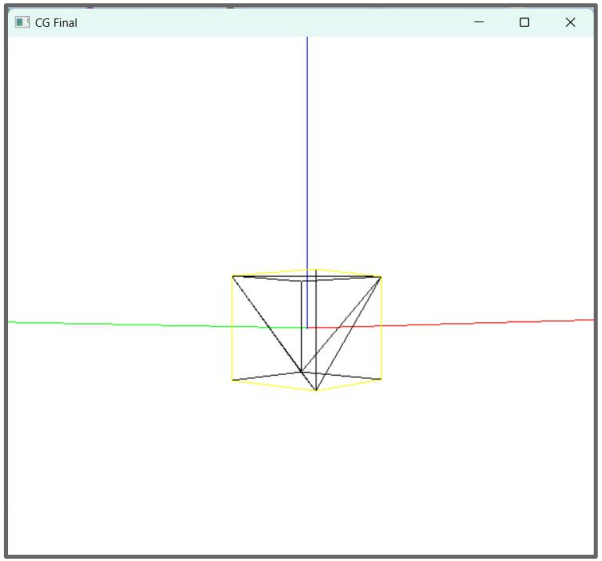
我們的程式遇到凹多邊形時，無法保留缺口

使用者介面比較



論文介面：比較看不出立體感





我們的介面：

1. 能提供更清楚的物體立體感。
2. 可在特定視角繪圖的同時，看到所生成物體不同角度的形狀。

困難與挑戰

PyOpengl安裝問題

- **glutInit() NullFunctionError**

  呼叫 “glutInit()” 時，收到 “NullFunctionError: Attempt to call an undefined function glutInit, check for bool(glutInit) before calling” 的錯誤訊息

- **原因**

  PyOpenGL中的 GLUT 或 FreeGLUT，不包含在 PyOpenGL 本身中

- **Windows11 CMD解決方法**

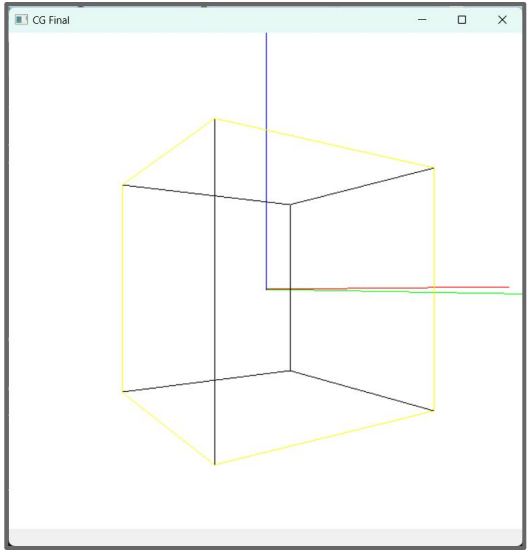
  下載

  1. PyOpenGL-3.1.6-cpPython版本-cpPython版本-win幾位元.whl
  2. PyOpenGL_accelerate-3.1.6-cpPython版本-cpPython版本-win幾位元.whl

（參考連結:https://s--nako-work.translate.goog/2021/02/trouble-shooting-nullfunctionerror-of-glutinit-in-opengl/?_x_tr_sl=en&_x_tr_tl=zh-TW&_x_tr_hl=zh-TW&_x_tr_pto=sc）

其他困難與挑戰

- CDT將凹多邊形變成凸多邊形:造成圖案與使用者給出的不同
- PyOpengl Shading:看不出立體感(解決)
- 將形成的物件投影到繪圖畫面(解決)
- 將繪圖介面上的點轉換到空間座標



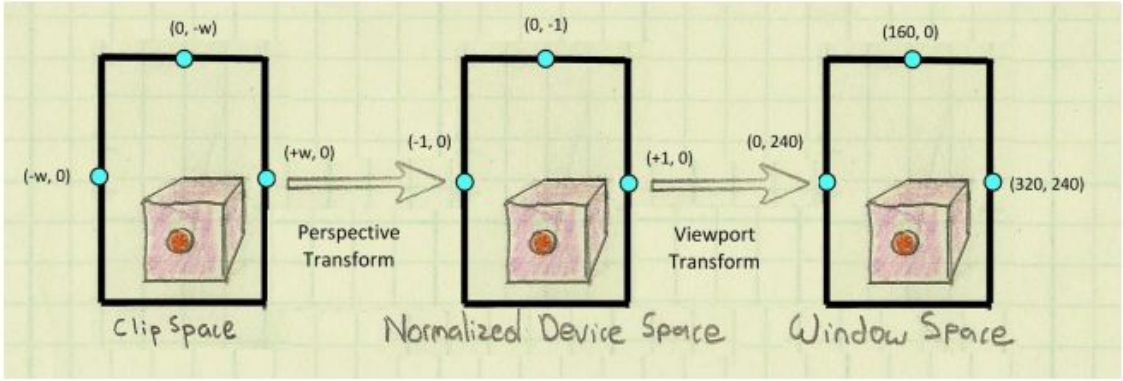


Image by Philp Rideout

Thanks