2IV10/2IV60 Computer Graphics

Examination, April 16 2013, 14:00 – 17:00

This examination consist of **four** questions with in total 16 subquestion. Each subquestion weighs equally. In all cases: **EXPLAIN YOUR ANSWER. Use sketches where needed to clarify your answer. Read first all questions completely.** If an algorithm is asked, then a description in steps or pseudo-code is expected, which is clear enough to be easily transferred to real code. Aim at compactness and clarity. Use additional functions and procedures if desired. Give from each function and procedure a short description of input and output. The use of the book, copies of slides, notes and other material is not allowed.

1 We consider some basic techniques for computer graphics.

a) What is a viewport in computer graphics terminology?

A viewport is a region of the screen that is used for showing graphics output, typically a rectangular area.

b) Give a criterion to distinguish convex and concave polygons.

Some characteristics of a convex polygon are that:

- All interior angles are < 180 degrees;
- All line segments between two interior points are completely inside the polygon;
- Given a line through an edge, all points of the polygon are either on this line or on the same side of the line;
- From each interior point, the complete boundary is visible.

These do not hold for concave polygons, and can hence be used to distinguish between them.

c) In the simplest model for transparency the color $I_{surface}$ of the surface and the color I_{back} of the background are blended into a perceived color I. Give a formula for I, assuming a transparency coefficient α in the range from 0 (opaque) to 1 (fully transparent).

In the simplest model, $I_{surface}$ and I_{back} and are weighted with $(1-\alpha)$ and α, and added up, i.e.,

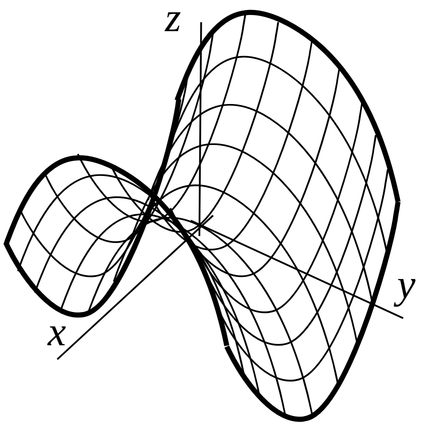
$I = (1-\alpha)\, I_{surface} + \alpha\, I_{back}$.

It is easy to mix up opacity and transparency. To check, substitute $\alpha = 0$ (opaque), which gives $I = I_{surface}$, hence we see the surface, which is correct. Also, $\alpha = 1$ (fully transparent) gives $I = I_{back}$, hence we see the only the background, which is correct again. Finally, if $I_{surface} = I_{back} = C$, we get $I = (1-\alpha)\, C + \alpha C = C$, hence if surface and background have the same color, transparency does not matter anymore.

d) What is the difference in the computation of light intensities between Phong shading and Gouraud shading?

In Gouraud shading *colors* of vertices are interpolated over polygons, in Phong shading *normals* of vertices are interpolated (followed by a shading calculation).

2 We consider a part of a hyperbolic surface, described by
$z = x^2 - y^2$ with $-1 \le x \le 1$ and $-1 \le y \le 1$.



a) Give a parametric description $S(u, v)$ and an implicit description $F(x, y, z)=0$ of this surface.

We take $x = u$ and $y = v$, and get $z = u^2 - v^2$. Hence,
$S(u,v) = (u, v, u^2 - v^2)$.
For the implicit equation, we rewrite the given equation to get
$z - x^2 + y^2 = 0$, hence $F(x, y, z) = z - x^2 + y^2$.

b) Derive a formula for a normal vector for a point on this surface, either using a parametric or an implicit description.

Using the parametric description, we get:

$$\begin{aligned} N(u,v) &= \frac{\partial S(u,v)}{\partial u} \times \frac{\partial S(u,v)}{\partial v} \\ &= (1,0,2u) \times (0,1,-2v) \\ &= (-2u, 2v, 1). \end{aligned}$$

Using the implicit equation, we get:

$$N(x,y,z) = \nabla F(x,y,z) = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z} \right) = (-2x, 2y, 1).$$

c) Calculate all intersection points of a line $P(t) = C + Vt$ with this surface, with $P = (P_x, P_y, P_z)$, $C = (C_x, C_y, C_z)$, and $V = (V_x, V_y, V_z)$.

A typical ray-tracing task. First, note that $P(t) = C + Vt = (C_x + V_x t, C_y + V_y t, C_z + V_z t)$. Next, we calculate for which value(s) of t we find intersections with the infinite surface. Substitution of $P(t)$ into the equation $z = x^2 - y^2$ gives

$$C_z + V_z t = (C_x + V_x t)^2 - (C_y + V_y t)^2.$$

We rewrite this to get a quadratic equation in t:

$$C_z + V_z t = C_x^2 - C_y^2 + (2C_x V_x - 2C_y V_y)t + (V_x^2 - V_y^2)t^2, \text{ or}$$

$$(V_x^2 - V_y^2)t^2 + (2C_x V_x - 2C_y V_y - V_z)t + (C_x^2 - C_y^2 - C_z) = 0.$$

Set $a = (V_x^2 - V_y^2)$, $b = (2C_x V_x - 2C_y V_y - V_z)$, and $c = (C_x^2 - C_y^2 - C_z)$

If $D = b^2 - 4ac < 0$, then there are no intersections. If $D > 0$, we get two values for t, using the well-known *abc* formula:

$$t_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

If $D = 0$ and $a \ne 0$, there is a single intersection for $t_1 = -b/(2a)$, provided that $a \ne 0$.
The case $D=0$ and $a = 0$ can occur, for instance for $C = (0, 0, 0)$ and $V = (1, -1, 0)$. Such a line lies completely in the unbounded surface.
If we find one or two intersection points, the final step is to check whether the point(s) $P(t_i)$ are located within the given points, i.e., the conditions

$$-1 \le C_x + V_x t_i \le 1 \text{ and } -1 \le C_y + V_y t_i \le 1$$

must be met.

d) Give a procedure to draw this surface, assuming a procedure *DrawTriangle*(*A*, *B*, *C*) is available.

For instance:

```
N = 20;       // number of steps per side
d = 1.0/N;    // stepsize

function Pnt(i, j): point;    // returns a point on the surface, specified by indices i and j (0...N)
begin
    Pnt.x = i*d; Pnt.y = j*d; Pnt.z = (i*i – j*j)*d*d;
end;

procedure DrawSurface;
begin
  for i = 0 to N–1 do
  for j = 0 to N–1 do
  begin
    P00 = Pnt(i,    j);          // Calculate points of a quad
    P10 = Pnt(i+1, j);
    P01 = Pnt(i,    j+1);
    P11 = Pnt(i+1, j+1);
  end;

  DrawTriangle(P00, P10, P11);   // Draw the quad with two triangles
  DrawTriangle(P00, P11, P01);
end;
```

Many alternatives are possible, for instance by storing and reusing points.

---

3 We want to draw an arc as shown in the figure. The arc starts in point $A = (a, 0)$, passes through point $B = (0, b)$, and ends in point $C = (–a, 0)$. At point A and C the arc is perpendicular to the x-axis, at point B the arc is perpendicular to the y-axis. We want to define the curve *parametrically* as $P(t)$, with $t \in [0, 1]$. We explore different options to define this arc. Indicate for each option if it is possible to define an arc that meets the requirements, and if not, explain why not; if yes, explain how this can be done and define $P(t)$ exactly. We consider:

a) Use of an ellipse (scaled circle);

This is possible: $P(t) = (a \cos(\pi t), b \sin(\pi t))$.

b) Use of a curve based on a cubic function $y = a_3x^3 + a_2x^2 + a_1x + a_0$;

This is not possible. At point A and C the tangent to the curve is vertical, and this implies that the derivate of y is infinite.

c) Use of a single quadratic Bézier segment $P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$; and

Again, not possible. A curve that starts at A and ends at C can be obtained by choosing $P_0 = (a, 0)$ and $P_2 = (–a, 0)$. To obtain vertical tangents at these points, we get $P_1 = (a, p)$ and $P_1 = (–a, p)$. These cannot be satisfied simultaneously. Or, equivalently, P_1 must be located at the crossing of the tangent lines at A and B, and if these lines are parallel, such a point cannot be found.

d) Use of a single cubic Bézier segment $P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$.

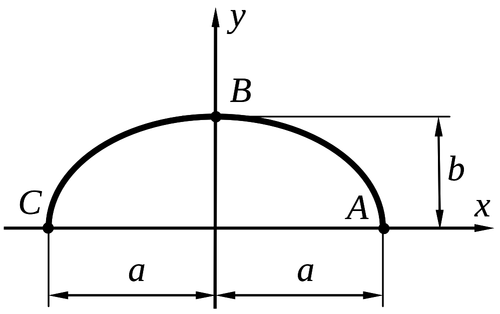
This is possible. A curve that starts at A and ends at C can be obtained by choosing $P_0 = (a, 0)$ and $P_3 = (-a, 0)$. To obtain vertical tangents at these points and aiming at a symmetric arc, we set $P_1 = (a, p)$ and $P_2 = (-a, p)$, where p is a constant to be determined.
Halfway, the arc should cross the y-axis with $y=b$, i.e.,

$$P_y(1/2) = b.$$

Substitution of the values picked for the control-points gives

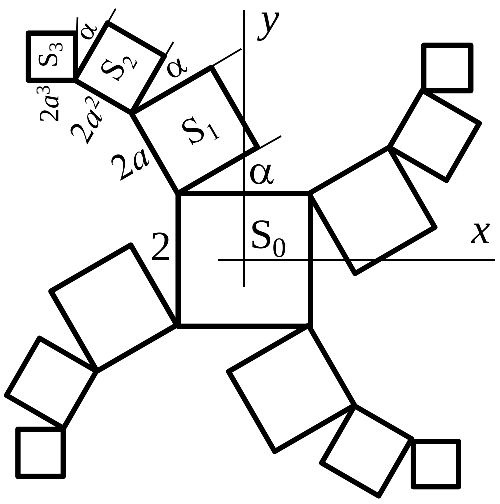
$$\begin{aligned} P_y(1/2) &= P_{0y}/8 + 3\,P_{1y}/8 + 3\,P_{2y}/8 + P_{3y}/8 \\ &= 3p/4. \end{aligned}$$

Hence, if we choose $p = 4b/3$ we obtain a cubic Bézier segment that meets the requirements.



---

4 We aim to draw the figure shown. In the center is a square S_0, centered on the origin, with size 2. The square S_1 has size $2a$, $a < 1$, its lower left corner coincides with the upper left corner of S_0, and S_1 is rotated over α degrees. This pattern is repeated, the size of a square S_{i+1} is a times the size of square S_i. On top of the other edges squares are positioned similarly.
We use a 3×3 homogenous transformation matrix M, such that a position A in global coordinates is related to a position B in local coordinates via $A=MB$. It may be assumed that $T(x,y)$ gives a translation matrix along the vector (x,y); that $R(\varphi)$ gives a rotation matrix of φ degrees around the origin; and that $S(s)$ gives a uniform scaling matrix with a scale factor s. The routine *DrawSquare*() draws a square in the local coordinate frame that is implicitly defined by the matrix M. In these local coordinates, the square that is drawn has size 2 and is centered on the origin.



a) Set M such that a call to *DrawSquare*() draws S_1, exactly according to the specification given and the figure.

We can transform S_0 to S_1 using the following steps, using global transformations:

1) $T(1, 1)$: Move the square such that the lower left corner is in the origin;
2) $R(\alpha)$: Rotate the square around the origin over α degrees;
3) $S(a)$: Scale the square with a factor a;
4) $T(-1, 1)$: Move the lower left corner (at the origin) to the upper left corner of the original square.

Using local transformations, we get:

1) $T(-1, 1)$: Move the center of the square to the upper right corner;
2) $S(a)$: Scale the square with a factor a;
3) $R(\alpha)$: Rotate the square around its origin over α degrees;
4) $T(1, 1)$: Move the square such that its lower left corner moved to its origin.

Note that only the order is reversed, the transformations are the same. Furthermore, the rotation and scaling can be interchanged. Based on these transformations, we get

$M = T(-1, 1)S(a)\, R(\alpha)T(1, 1)$.

b) Suppose M has been set such that S_i has just been drawn with a call *DrawSquare*(). Update M to draw S_{i+1} with yet another call to *DrawSquare*().

Each time a new square is added, the same transformation is applied again. This can be seen if we consider S_i and check which transformations are needed (in local coordinates) to get S_{i+1} . Hence:

$M' = M\, T(-1, 1)S(a)\, R(\alpha)T(1, 1)$.

But also

$M' = T(-1, 1)S(a)\, R(\alpha)T(1, 1)\, M$.

gives the desired result. In general, the transformation M_i for S_i is given by

$M_i = (T(-1, 1)S(a)\, R(\alpha)T(1, 1))^i$

c) It is desired that square S_n has size p and is rotated over β degrees in total. How to set a and α to get this effect?

The total rotation must be equal to β, the figure shows that this is equal to $n\alpha$. Hence, $\alpha = \beta / n$.

The size must be equal to p, repeated scaling gives a size equal to $2a^n$. Hence, $a = (p/2)^{1/n}$.

d) Give a procedure to draw the complete figure, including all four arms, where each arm consists of n squares.

For instance:

```
procedure DrawFigure($n$);
begin
    $M = I$;                        // set $M$ to the identity matrix
    DrawSquare();                   // draw $S_0$
    $P = T(-1, 1)S(a)R(\alpha)T(1, 1)$;   // Calculate and store the  basic transformation step

    for $i$ = 1 to $n$ do  // for all levels…
    begin
        $M = MP$;         // adapt the transformation for the next level squares
        for $j$ = 1 to 4 do  //  for all arms…
        begin
            DrawSquare();   // draw a square
            $M = R(90)M$;     // apply a global rotation to shift to the next arm
        end;
        // Note: after for rotations over 90 degrees, $M$ is back to its starting position
    end;
end;
```

Alternatively, the loops for levels and arms can be interchanged, for instance:

```
procedure DrawFigure($n$);
begin
    $M = I$;                                // set $M$ to the identity matrix
    DrawSquare();                           // draw $S_0$
    $P = T(-1, 1)S(a)R(\alpha)T(1, 1)$;     // Calculate and store the  basic transformation step

    for $i$ = 0 to 3 do  // for all arms…
    begin
       $M = R(i*90)$;                       // set $M$ to a rotation of 0, 90, 180, 270 degrees
        for $j$ = 1 to $n$ do  //  for all levels…
        begin
            $M = MP$;            // adapt the transformation for the next level square
            DrawSquare();   // draw a square
        end;
    end;
end;
```