Drawing the cube

1. A new package for drawing in 3D

This package, which iscalled **`draw3d.inc`**, replaces **`projection3d.inc`** and extends it quite a bit.

Routine: **surface-transform**
Arguments: **`A s`**
Returns: The surface s transformed by the affine transformation A

Routine: **is-visible-in-projection**
Arguments: **`f`**
Returns: **`true`** or **`false`**

Routine: **is-visible-in-perspective**
Arguments: **`f`**
Returns: **`true`** or **`false`**

These two apply to the transformed face f.

Routine: **projection**
Arguments: **`[x y z]`**
Returns: **`[x y]`**

Routine: **perspective**
Arguments: **`[x y z]`**
Returns: **`[-x/z -y/z]`**

Routine: **shading**
Arguments: A 3D vector
Returns: A number g representing the shade of grey to use

Routine: **mksurface**
Arguments: **`pars /f [si ti] [sf tf] [ns nf]`**
Returns: The surface parametrized by f over the range indicated

Here f is a procedure with two arguments, an array of parameters and a 2D vector (s,t). It returns a 3D vector $f(s,t)$, where the calculation of f may use the elements of the array. It creates the surface as a collection of $n_s \times n_t$ rectangles with corners at $f(s,t)$, $f(s+ds,t)$, $f(s+ds,t+dt)$, $f(s,t+dt)$, where $dt = t_f - t_i/n_t$, $ds = s_f - s_i/n_s$, and the parameters s and t range over the rectangle $[s_i, s_f] \times [t_i, t_f]$.

```
% renders surfaces in 3d

% needs matrix3d.inc

% a surface is an array of faces
% a face = [ polygon + normal ]
% a polygon = array of vertices

% affine transformation A + a surface s => transformed surface

/surface-transform {
4 dict begin
/s exch def
[
  s {
```

```
  /f exch def
  /p f 0 get def
  /n f 1 get def
  [
    [
      p {
        A exch affine-transform-3d
      } forall
    ]
    A 0 get n transform-3d
  ]
  } forall
]
end
} def

% face

/is-visible-in-projection {
1 get 2 get 0 gt {
  true
} {
  false
} ifelse
} def

% face

/is-visible-in-perspective {
1 dict begin
/f exch def
  f 0 get 0 get
  f 1 get
  dot-product-3d
0 lt {
  true
} {
  false
} ifelse
end
} def

% [x y z] => [x y]

/projection {
1 dict begin
/v exch def
[v 0 get v 1 get]
end
} def

% [x y z] => [-x/z -y/z]
```

```
/perspective {
2 dict begin
/v exch def
/z v 2 get neg def
[v 0 get z div v 1 get z div]
end
} def

% ------------------------------------------------

/light-source [-1 1 0.5] normalize-3d def

% normal

/minshade 0.05 def
/maxshade 0.95 def
/shadefactor maxshade minshade sub 4 div def

/shading {
2 dict begin
/n exch def
/d n light-source dot-product-3d
def
maxshade minshade add 0.5 mul
maxshade minshade sub 0.5 mul
d mul add
end
} def

% pars /f [si ti] [tf sf] [ns nt] => surface parametrized by f
% and range [si sf] x [ti tf] - get ns x nt rectangles
% f:  pars [s t] => f(s, t)

/mksurface {
16 dict begin
aload pop
/nt exch def
/ns exch def
aload pop
/tf exch def
/sf exch def
aload pop
/ti exch def
/si exch def
/f exch cvx def
/pars exch def
/sinc sf si sub ns div def
/tinc tf ti sub nt div def

[
  /s si def
```

```
  ns {
    /t ti def
    nt {
[s t] ==
      % make the f(s, t) face
      /ll pars [s         t        ] f def
      /lr pars [s sinc add t        ] f def
      /ur pars [s sinc add t tinc add] f def
      /ul pars [s         t tinc add] f def
      [
      [
        ll lr ur ul ll
      ]
      /n lr ll vector-sub-3d
        ul ll vector-sub-3d
        cross-product
      def
      /r n vector-length-3d def
      r 0 ne {
        /n n 1 r div vector-scale-3d def
      } {
        /n ur lr vector-sub-3d
          ur ul vector-sub-3d
          cross-product
        def
        /r n vector-length-3d def
        r 0 ne {
          /n n 1 r div vector-scale-3d def
        } {
          /n [0 0 1] def
        } ifelse
      } ifelse
      n ]
      /t t tinc add def
    } repeat
    /s s sinc add def
  } repeat

]

end
} def
```

2. Using it to draw the cube

In this section, parts of the package above are used to do one of the problems on animating rotation of a cube (in perspective, but without shading). The other problems are minor modifications of this. Note the new PostScript command **`forall`** which loops over elements of an array, putting each element in turn on the stack at the beginning of the loop. We could have used a **`for`** loop as well.

Note also **`1 setlinecap`** and **`1 setlinejoin`**, which smooth out the corners of the cube.

```
%!

% draws a picture of a unit cube rotating in space

/pagesetup { 72 dup scale
4 5 translate
/d 5 def
0.01 d div setlinewidth
d dup scale
1 setlinecap
1 setlinejoin } def

(matrix3d.inc) run
(draw3d.inc) run
(cube.inc) run

/theta 0 def
/inc 10 def

36 {

gsave
pagesetup

/A [ [1 1 0] theta rotation-matrix-3d [0 0 d neg] ] def
/theta theta inc add def

A cube surface-transform {
  /f exch def
  f is-visible-in-perspective {
    /p f 0 get def
    newpath
    p 0 get perspective aload pop moveto
    /n p length 1 sub def
    1 1 n {
      /i exch def
      p i get perspective aload pop lineto
    } for
    stroke
  } if
} forall % faces on the surface

grestore
showpage

} repeat
```