casting shadows on flat surfaces

By Thant Tessman

The ability to cast natural-looking shadows is critical to the development of realistic computer-generated images. Happily, in the case of SGI systems, the Geometry Pipeline can be used to draw an object flattened against an arbitrary plane. If the flattened object is drawn in black, it can be used as the original object's shadow. Demos such as *flight* and *insect* are examples of programs that include shadows produced in this way.

The easiest way to flatten an object is to use the **scale** function. If the ground is the plane formed by the x and z axes (y=0), then:

scale(1.0, 0.0, 1.0)

could be used to create a shadow from a light shining straight down the y axis. Displayed in Figure 1 is a program fragment usable for producing shadows.

```
zbuffer(FALSE);          /* don't bother z-buffering ground or shadow */
                         /* unless something needs to intersect it */

lookat(vx, vy, vz, px, py, pz, twist);
draw_ground();

setpattern(halftone);    /* this is so the shadow isn't solid black but */
RGBcolor(0, 0, 0);       /* only darkens every other pixel */
                         /* (assumes halftone defined already) */

pushmatrix();
scale(1.0, 0.0, 1.0);    /* shadow transformation */

position_object();       /* a set of rotates and translates that's normally */
                         /* used to position and orient the object */

draw_object_shadow();    /* the shadow version of the object should be */
                         /* the same as the object, only all black */
                         /* (no lighting or color) */

popmatrix();

setpattern(0);
zbuffer(TRUE);

position_object();       /* position and orient the object just like before, */
                         /* except this time without the shadow transform */

draw_object();           /* draw shading lit from above */
```

Figure 1: *A program fragment for producing shadows.*

(By archaic convention, the y axis, not the z, points up. To use the **lookat** function with data that assumes z is up, **lookat** should be followed by **rotate (-900, 'x')** ; this will make z become y, and y become -z. The techniques described here can be generalized for other directions, but that is left as an exercise for the reader.)

The **scale** function by itself is good only for shadows where the light is parallel to one of the major axes. A set of rotations and scales can be used to cast a shadow from an arbitrary direction, but it is much easier to build a single transformation matrix and use the **multmatrix** function. The hand-built matrix can then be inserted into the code fragment, replacing the **scale** function.

As shown in Figure 2, the equations for casting a shadow to the ground at $y = 0$ (assuming an infinite light source) are:

$$X_{new} = X_{old} + Y_{old} \frac{lx}{ly}$$

$$Y_{new} = 0$$

where lx and ly are the components of the light vector. The equation for the third dimension is:

$$Z_{new} = Z_{old} + Y_{old} \frac{lz}{ly}$$

The transformation matrix built from these equations follows:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{lx}{ly} & 0 & \frac{lz}{ly} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

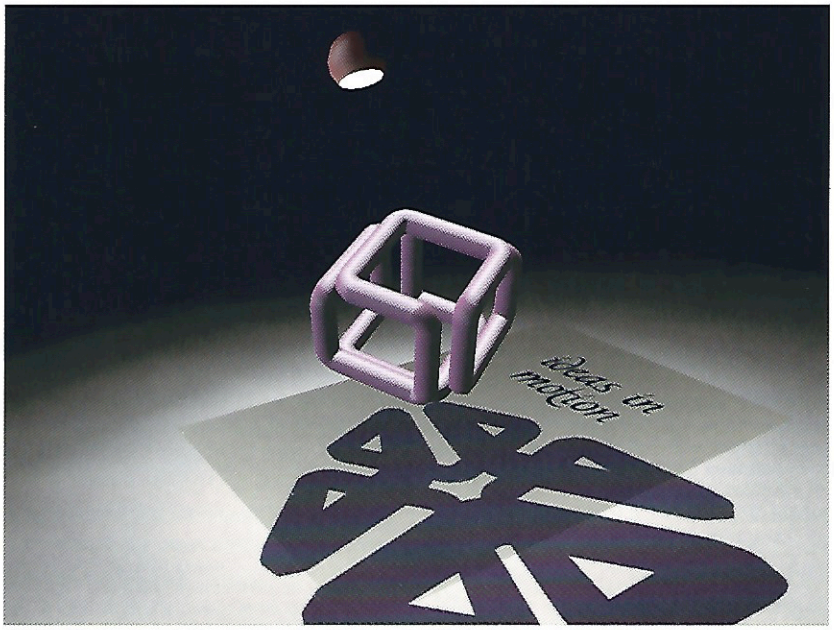
Note that if *lx* and *lz* are zero, (*i.e.* if the light is shining straight down), the matrix will do the same thing **scale(1.0, 0.0, 1.0)** would.

This matrix and the one created by the **scale** function produce shadows as if the light were infinite. In this sense, they are orthogonal. A matrix very similar to the one created by the **perspective** function will generate shadows from a local light source (see Figure 3).

Here, *lx* and *ly* are the absolute coordinates of the light, because the position of the point relative to the light must be used. Just as before, the ground is at *y*=0:

$$Y_{new} = 0$$

By similar triangles:



A still from "Ideas in Motion: The Movie", written in C and generated in real time (approximately eight to fourteen frames per second, depending on window size) using an IRIS 4D/70GT.

$$\frac{lx - X_{new}}{ly} = \frac{lx - X_{old}}{ly - Y_{old}}$$

where *ly* really is *ly*-*Ynew* but *Ynew* is zero.

The fourth column in transformation matrices (the *w* column) is used for the perspective divide. A matrix really transforms *x*, *y*, *z*, and *w*, where *w* usually is 1.

The resulting *x*, *y*, *z*, and *w* are normalized to make *w*=1 (by dividing

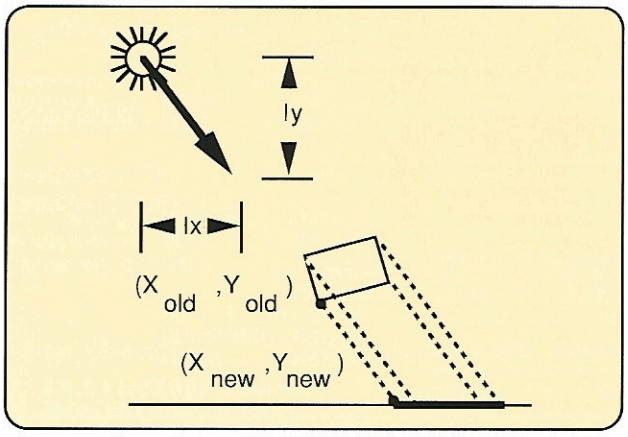


Figure 2: *Shadow from an infinite light source.*

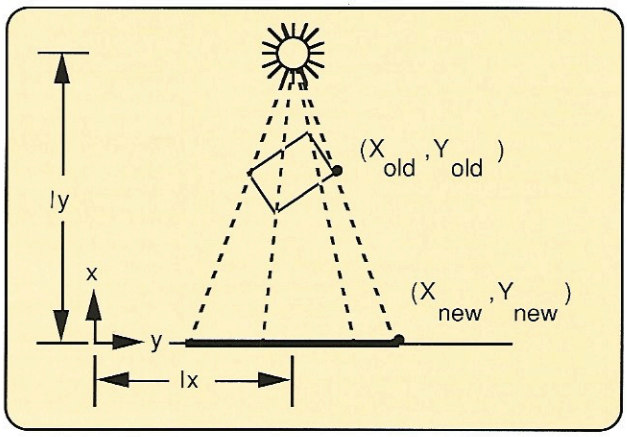


Figure 3: *Shadow from a local light source.*

each of them by w). Solving for $Xnew$:

$$X_{new} = \frac{ly\,X_{old} - lx\,Y_{old}}{ly - Y_{old}}$$

For constructing the matrix, W_{new} is $ly - Y_{old}$. Similarly for $Znew$:

$$Z_{new} = \frac{ly\,Z_{old} - lz\,Y_{old}}{ly - Y_{old}}$$

The transformation matrix built from these equations is:

$$\begin{bmatrix} ly & 0 & 0 & 0 \\ -lx & 0 & -lz & -1 \\ 0 & 0 & ly & 0 \\ 0 & 0 & 0 & ly \end{bmatrix}$$

Although this technique is very specialized, it also is very quick. In fact, the overhead is low enough that there is no reason *not* to rebuild a matrix for each frame to allow for a moving light source. Combined with hardware-assisted lighting, the dynamic images that result can look surprisingly realistic.

Thant Tessman is a Silicon Graphics systems engineer working in the company's Advanced Systems Division.

For more information on transformation matrices, see Mathematical Elements for Computer Graphics by David F. Rogers and J. Alan Adams (McGraw-Hill, 1976). Also see Chapter 4, "Coordinate Transformations", and Appendix C: "Transformation Matrices" in the Graphics Library User's Guide, Volumes I and II.

SILICON GRAPHICS, INC. EDUCATION CENTER
COURSE CALENDAR
(Through June 1989)

COURSE	LOCATION*	
3000 SERIES COURSES	WEC	EEC
GRAPHICS I 4.5 days	Feb 27, 1989 Jun 26, 1989	not available
SYSTEM ADMINISTRATION 4.5 days	Apr 24, 1989	not available
SYSTEM MAINTENANCE 10.0 days	May 15, 1989	not available
4D SERIES COURSES		
GRAPHICS I 4.5 days	Mar 13, 1989 May 1, 1989 Jun 12, 1989	Feb 27, 1989 Apr 10, 1989 Jun 5,1989
ADVANCED GRAPHICS 3.5 days	Mar 20, 1989 Jun 19, 1989	Apr 17, 1989
PARALLEL PROGRAMMING	Mar 27, 1989	May 22, 1989
SYSTEM ACCELERATOR 4.5 days	Apr 3, 1989 May 8, 1989	Mar 6, 1989 May 1, 1989 Jun 12, 1989
SYSTEM ADMINISTRATION 4.5 days	May 15, 1989	Mar 13, 1989 Jun 19, 1989
NETWORK ADMINISTRATION 4.5 days	Mar 6, 1989 Jun 5, 1989	Apr 3, 1989
SYSTEM MAINTENANCE 10.0 days	Apr 10, 1989	May 8, 1989
PERSONAL IRIS SYSTEM MAINTENANCE 3.5 days	Mar 27, 1989 May 8, 1989	not available

KEY:
WEC--*SGI Western Education Center, Mountain View, CA*
EEC--*Eastern Education Center, SGI Federal, Bethesda, MD*

*The SGI Education Center reserves the right to cancel classes due to insufficient enrollment.