Scene Format

Paul Heckbert
27 Feb. 1996

Scene format is a simple file format for describing 3-D scenes for computer graphics. It is meant to be useful for a variety of renderers, including z-buffer, ray tracing, and radiosity algorithms. Scene format borrows some ideas from the Renderman Interface Bytestream format, from Postscript, and from OpenGL.

In scene format, a 3-D scene is built up using geometry commands such as `sphere` and `poly3`, which instance geometric shapes. The coordinate system, material attributes, and lighting for each shape come from the *graphics state* when the geometry command occurs. The graphics state is local; it can vary from line to line. The other portion of the state, the *global state*, is set at the beginning of a scene file, before any geometry commands, and stays constant throughout the scene. The global state includes the camera transformation and the background color.

Syntax

Commands consist of a keyword followed by whitespace-separated parameters. Parameters are floating point numbers, unless specified otherwise. There is no command termination character (such as ';' or newline). Whitespace is one or more space, tab, newline, or comment. A comment begins with the character # and extends to the next newline.

Geometry Commands

The following commands create instances of geometric primitives at the given location in the current coordinate system.

`sphere` $x\ y\ z$ *rad*
: Create a sphere with center (x, y, z) and radius *rad*.

`poly3` $n\ \ x_1\ y_1\ z_1\ \ x_2\ y_2\ z_2\ \ \ldots\ \ x_n\ y_n\ z_n$
: Create a polygon with n vertices $(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)$. The polygon can be assumed to be planar and convex. As a convention for backface testing, polygons should be counterclockwise when viewed from the "outside".

`poly2` $n\ \ x_1\ y_1\ \ x_2\ y_2\ \ \ldots\ \ x_n\ y_n$
: Like `poly3`, except all vertices are assumed to be in the $z = 0$ plane.

Transformation Commands

The following commands modify the current transformation by performing a transformation to the local coordinate system. To transform a point from its local coordinate system (object space) to the global coordinate system (world space or screen space), the transformations are applied in the reverse of the order of commands in the scene file.

`translate` $t_x\ t_y\ t_z$
: Translate by the vector (t_x, t_y, t_z).

`rotate` *axis angle*
: Rotate *angle* degrees about the specified axis (*axis*=`x`, `y`, or `z`), according to the right hand rule.

`rotgen` $a_x\ a_y\ a_z$ *angle*
: Rotate *angle* degrees about the axis (a_x, a_y, a_z) according to the right hand rule. The length of the axis is irrelevant unless it is zero, in which case the command has no effect.

`scale` s_x s_y s_z

Scale x, y, and z by s_x, s_y, and s_z, respectively. If $s_x = s_y = s_z$ then the scale operation is called "uniform", otherwise it is called "nonuniform".

`push`

Push current transformation on stack, saving it.

`pop`

Pop current transformation off stack, restoring it to the previously-pushed state.

Camera Transformation Commands

Camera transformations define the sequence of transformations between world space and screen space. Screen space is the initial coordinate system for the first scene file. Camera transformations should all come before the first geometry command. The last camera command should be `world_space`.

`screensize` *width height depth*

A convenience function to set the width, height, and depth of screen space (the viewport) in pixels. It is intended to be used to map normalized screen space to screen space. Precisely, this command scales and translates like so:

`scale` *width*/2 $-$*height*/2 *depth*/2
`translate` 1 -1 0

Screen space y is assumed to point down, hence the negative y-scale. As a side effect, this command sets the width, height, and depth which will typically be used by a renderer to set display resolution.

`xyzrange` *xmin xmax ymin ymax zmin zmax*

A convenience function for parallel (not perspective) camera transformation. It is intended to be used to map the specified xyz window of eye space to normalized screen space, i.e. to $[-1,1]$ in x, y, and z. Precisely, this command scales and translates like so:

`scale` 2/(*xmax*$-$*xmin*) 2/(*ymax*$-$*ymin*) 2/(*zmax*$-$*zmin*)
`translate` $-$(*xmin*+*xmax*)/2 $-$(*ymin*+*ymax*)/2 $-$(*zmin*+*zmax*)/2

`zrange` *zmin zmax*

Scales and translates z only. Useful for z-buffering. Intended to be used along with `persp` to map eye space to normalized screen space, placing the near and far clipping planes at *zmin* and *zmax*, respectively. This command followed by a `persp` command maps $z =$ *zmin* to $z = -1$, and $z =$ *zmax* to $z = 1$. Precisely, it translates and scales like so:

`translate` 0 0 (*zmax*+*zmin*)/(*zmax*$-$*zmin*)
`scale` 1 1 $-2/(1/$*zmin*$-1/$*zmax*$)$

`persp` *fov aspectratio*

Perform a perspective transformation with the eye at (0,0,0) looking in the +z direction, with horizontal field of view *fov* degrees and aspect ratio (ratio of picture width to height) of *aspectratio*. Points within this pyramid are mapped to the normalized screen space range of $[-1,1]$ in x and y.

`lookat` f_x f_y f_z t_x t_y t_z u_x u_y u_z

A convenience function to position and orient the camera: it places the camera at the "from-point" (f_x, f_y, f_z) looking toward the "to-point" (t_x, t_y, t_z), oriented so that the "up vector" (u_x, u_y, u_z) is up. This command should typically be preceded by `scale 1 1 -1`. Precisely, this command translates by $-\vec{f}$, then rotates such that $\vec{f}-\vec{t}$ is mapped to the $-z$ axis, and $\vec{u}$ is mapped to the yz plane (as close as possible to the y axis).

`world_space`

States that the current coordinate system is world space.

If no camera commands are given, then world space and screen space will be identical. For a perspective view, the typical camera command sequence is `screensize`, `persp`, `scale 1 1 -1`, `lookat`, `world_space`. For a parallel view, replace `persp` with `xyzrange`. Standard transformation commands such as `translate` and `rotate` can also be used in a camera definition. If the range of z-values is important (as when z-buffering), insert a `zrange` just before `persp`, and set $depth$ appropriately (for 16-bit z, use $depth = 65535$). If z is unimportant, then $depth = 2$ is recommended. Avoid using $depth = 0$, since that creates a degenerate transformation (a singular matrix).

Material Commands

Material commands redefine the current material, overriding the previous setting.

`diffspec` $r\ g\ b$ ***kdiffrefl kspecrefl kspectran*** $expon\ index$
: Current material is some combination of reflective and transmissive, with color (r, g, b), diffuse reflectance, specular reflectance, and specular transmittance of ***kdiffrefl***, ***kspecrefl***, and ***kspectran***, respectively, Phong exponent $expon$ (related to roughness), and index of refraction $index$. Typically, $0 \leq r, g, b \leq 1$, $0 \leq$ ***kdiffrefl***$+$***kspecrefl***$+$***kspectran*** ≤ 1, $10 \leq expon \leq 200$, and $1 \leq index \leq 2.5$.

`diffuse` $r\ g\ b$ ***kdiffrefl***
: Current material is opaque and diffuse with reflectance ***kdiffrefl*** and color (r, g, b).

`emissive` $r\ g\ b$ ***kemission***
: Current material is emissive (it emits light) and diffuse with emission coefficient ***kemission*** and color (r, g, b). This is useful for radiosity.

Light Commands

`ambient` $r\ g\ b$ ***kambient***
: Ambient light has color ***kambient***$\times (r, g, b)$. The factor ***kambient*** is a multiplier to facilitate brightening and dimming of lights without altering their hue. Default: black.

`pointlight` $x\ y\ z\ \ r\ g\ b$ ***kemission***
: Create a point light source at (x, y, z) with color ***kemission***$\times (r, g, b)$.

Miscellaneous Commands

`gpush`
: Push entire graphics state on stack, saving it. This includes the current transformation, the current material, and the current set of lights.

`gpop`
: Pop graphics state off stack, restoring it to previously `gpush`-ed state (doing `pop`s, if necessary). The scopes of `gpush`–`gpop` and `push`–`pop` can be nested, but cannot overlap.

`background` $r\ g\ b$ ***kbg***
: Background has color ***kbg***$\times (r, g, b)$. The background color is part of the global state, so a `background` command should only occur before the first geometry command. Default: black.