ADSODA

Arbitrary-Dimensional Solid Object Display Algorithm

by Greg Ferrar

Introduction

ADSODA is an algorithm for displaying solid objects of arbitary dimension. It includes *n*-dimensional lighting, *n*-dimensional hidden solid removal, and *n*-dimensional orthographic projection to *n-1* dimensions.

Motivation

Traditional high-speed rendering packages use one-dimensional and two-dimensional solid objects (line segments and polygons) for their internal scene representations, even when the scene is of three or more dimensions. For instance, fast three-dimensional renderers generally render objects as polygons, which are two-dimensional, rather than using fully three-dimensional objects like spheres or polyhedra. In three dimensions, this is not generally a problem, because full realism can be achieved using polygons. However, in higher dimensions, polygons no longer suffice-- just as 0D points 1D line segments are not sufficient to render realistic 3D scenes into two dimensions, 1D line segments and 2D polygons are not sufficient to render realistic 4D scenes into three dimensions.

Furthermore, the introduction of fully solid *n*-dimensional objects brings with it the problem of hidden solid removal; naively drawing the projections of all solids in a space is insufficient, because the projections of an object will obscure objects in front if it, if it is drawn in the wrong order. This problem can be effectively solved in three dimensions using a z-buffer, or by drawing back-to-front. For *n* dimensions, however, a z-buffer required extraordinary amounts of memory, and back-to-front ordering works only if rendering is being done immediately after projection into some sort of a "bitmap", where individual pixels or voxels can overwrite the voxels behind them. In arbitrary *n*-dimensional rendering, this condition does not hold. For instance, two projections are required to render a 4-dimensional object on a 2-dimensional screen, and the intermediate 3-dimensional projection cannot be a "bitmap"; it must be a mathematical scene representation, so that it can be used to project to 2D.

A Solution

ADSODA addresses these issues by using an *n*-dimensional representation for a solid. which fully describes the *n*-dimensional volume occupied by the solid. This representation, a simplified form of CSG (constructive solid geometry), provides some operations, like object intersections, differences, and slicings, quite efficiently. Using these operations, it implements *n*-dimensional hidden solid removal (the natural extension of 3-dimensional hidden surface removal) in the source space, clipping out hidden parts of the scene from the source space before projecting. With this done, drawing can be done in arbitary order; since there is no overlap of the object projections, all possible orderings result in the same projected space, and the projected space objects to not intersect.

To render the *n*-dimensional solid realistically in *(n-1)*-space, ADSODA implements several other basic rendering algorithms. First, it use a simple lighting model which is highly efficient, due to the inclusion of normal vectors in the natural representation of solids. Second, it contains an orthographic projection algorithm for mapping *n*-spaces to *(n-1)*-spaces. Finally, it contains algorithms for rendering 2-dimensional and 3-dimensional solids using OpenGL. These algorithms combine to support the rendering of arbitrary dimensional solids with lighting effects in all dimensions, and hidden solid removal in all dimensions.

Due to the relative simplicity of the representation and the algorithms, ADSODA can render simple animated four-dimensional scenes in real time.

Solid Representation

ADSODA represents solid *n*-dimensional solids as the intersection of a set of halfspaces. For instance, it would represent the two-dimensional solid in **Figure 1a** as the intersection of four halfplanes in **Figure 1b**, **Figure 1c**, **Figure 1d**, and **Figure 1e** (**Figure 1f**). The halfspaces are represented by their equations, which can also be thought of as their normal vectors, plus a constant.

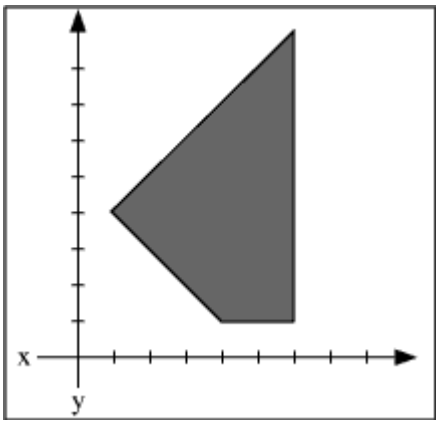
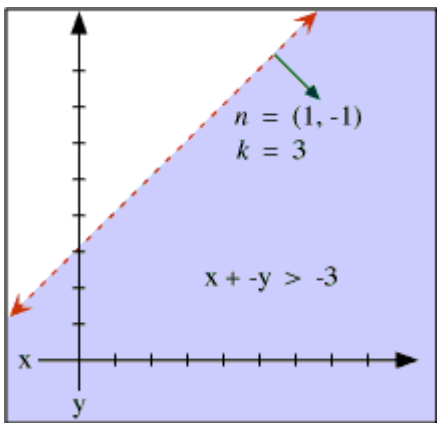


Figure 1a



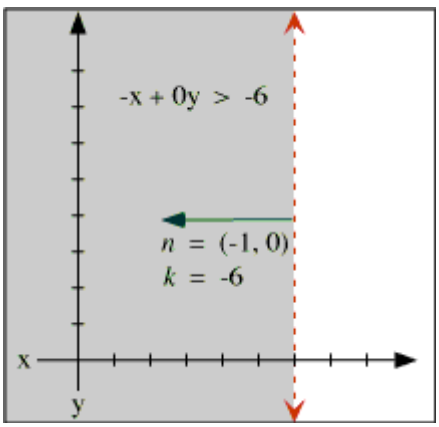
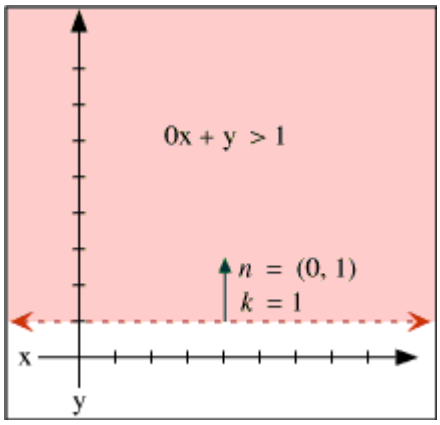


Figure 1b

Figure 1c



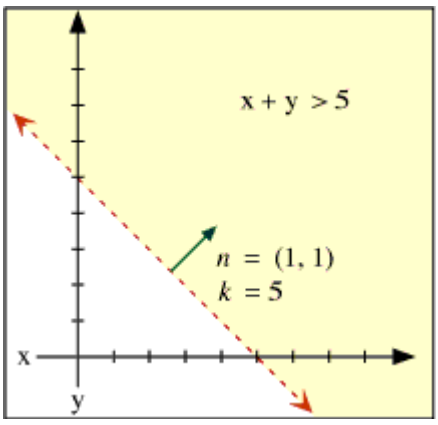


Figure 1d

Figure 1e

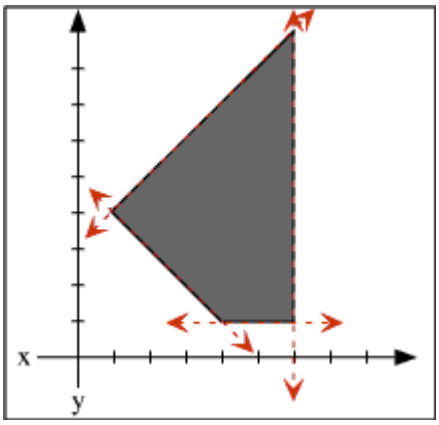


Figure 1f

This representation extends naturally into *n* dimensions. Operations involving slicing are trivial; to slice a solid along an arbitrary line, it is necessary only to add another halfspace to its representation (**Figures 2a** and **2b**).

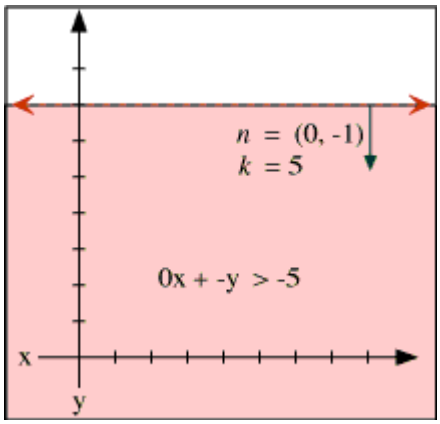


Figure 2a

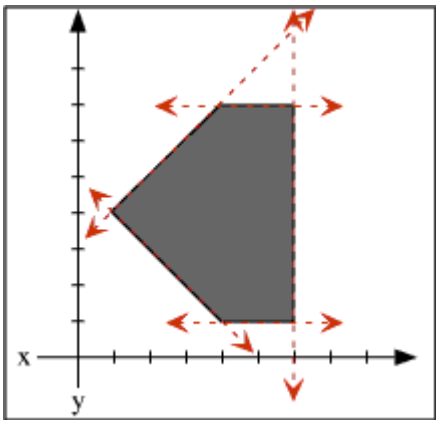


Figure 2b

Lighting

The solid representation model makes lighting fast and simple. The normal vectors of the faces of the solids being lit are built into the representations of the solids (each face of a solid is represented by its equation, as described above, and the non-constant coefficients of the equation are exactly the components of the normal vector). ADSODA uses a very simple lighting algorithm; it computes the dot product of the normalized normal vector with the normalized eye vector, and uses this as the brightness.

Hidden Solid Removal

Before an *n*-dimensional space can be projected onto *(n-1)* dimensions for display, hidden solid removal must take place. This removes all solids in the space which are obscured by other solids, as seen from the eye point. After hidden solid removal occurs, all solids in the space can be projected directly onto the lower dimensional space; hidden solid removal ensures that no "collisions" will occur (that in the lower dimensional space, no two projected solids will intersect). In the case of a 3D to 2D projection, for instance, this means that the resulting two-dimensional space can be immediately drawn on a screen without any regard for the order polygons are drawn in; there will be no overlap, so the end result of any order or rendering will be the same.

The first step in ADSODA's hidden solid removal algorithm is to compute the *silhouettes* of all the solids in the space. The *silhouette* of an *n*-dimensional solid is the "outline" of the solid, as seen from the observer. **Figure 3** shows a sample two-dimensional space with three solids (**Figure 3a**), with the silhouettes of the three solids. The observer in this example is two-dimensional, and is looking from the negative *y* direction. The silhouette is an *(n-1)*-dimensional solid, and is represented in the same way all *(n-1)*-dimensional solids are represented in ADSODA (as the intersection of a set of halfspaces; in **Figure 3**, each silhouette is represented as the intersection of two half-lines).

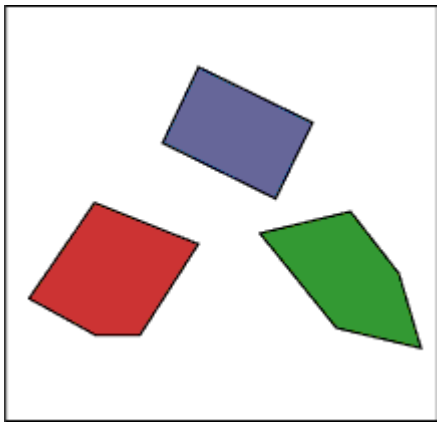


Figure 3a

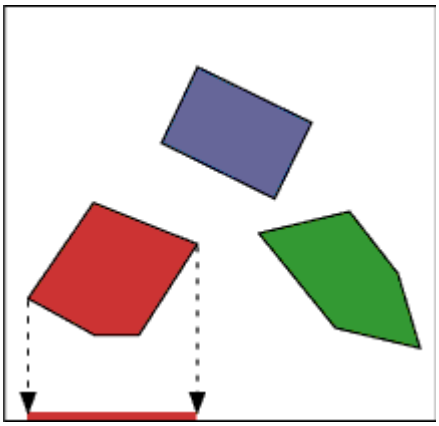


Figure 3b

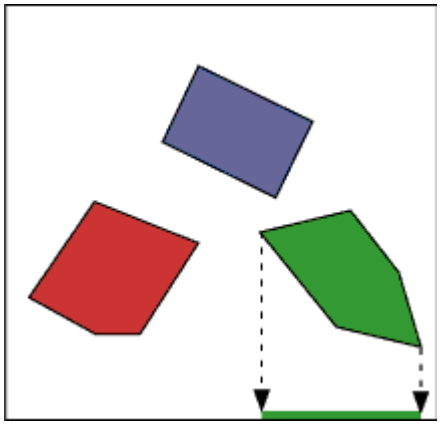


Figure 3c

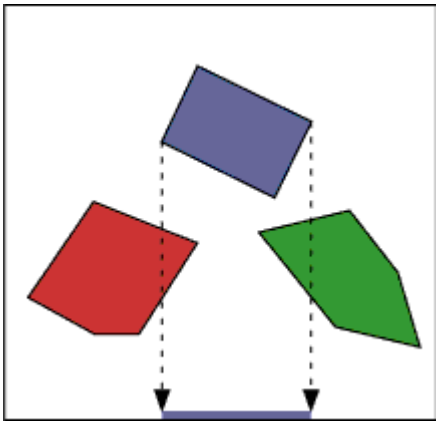


Figure 3d

To compute the silhouette of a solid, ADSODA finds every pair of adjacent faces of the solid for which one face is a frontface (facing towards the viewer) and the other face is a backface (facing away from the viewer). Frontfaces and backfaces can be easily determined by checking the sign of the dot product of viewing vector with the face normal vector. Given two such adjacent faces, ADSODA finds the intersection between the faces (which is an *(n-2)*-space), and projects it to the destination space. For instance in **Figure 3c**, the lower right corner of the green solid is the intersection of a frontface and a backface, and is therefore a border of the silhouette; the leftmost point of the green solid is the intersection of a frontface and a backface, and is therefore also a border of the silhouette; there are no other adjacent frontface/backface pairs, so there are no other borders to the silhouette.

For an *n*-dimensional solid, the faces of the solid are *(n-1)*-dimensional solids imbedded in *n*-space (line segments, in the case of **Figure 3**); the boundaries of the silhouette are *(n-2)*-spaces imbedded in the *(n-1)*-dimensional projection space (points, in the case of **Figure 3**).

As another example, consider the case of a cube in 3-space (**Figure 4**). To find the silhouette of this cube, ADSODA finds every edge between a frontface and a backface, and projects the edge (which is a line in 3-space) onto the destination plane. The result is a halfplane, which, combined with other halfplanes found in the same manner (there are six such halfplanes), represents the 2-dimensional silhouette solid using ADSODA's solid representation method.

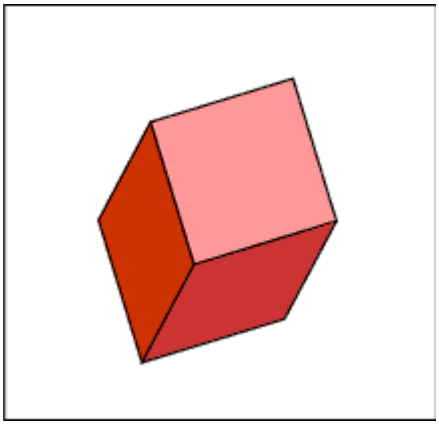


Figure 4a

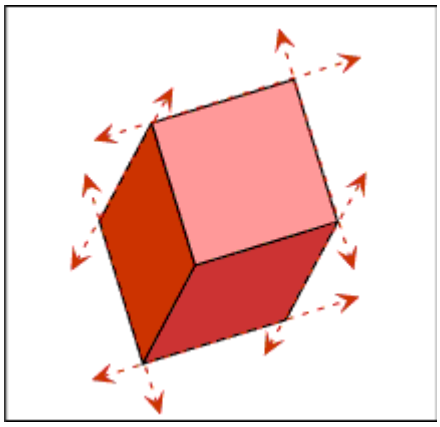


Figure 4b

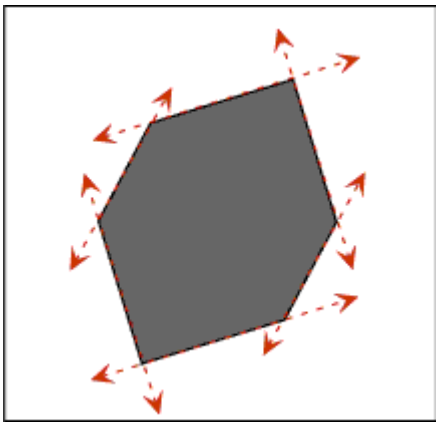


Figure 4c

Finding which faces are adjacent is non-trivial; this information is not present in the basic (intersection of halfspaces) representation. It must be computed in a separate step. ADSODA determines this information by first finding all corners of the solid, and then using them to find the adjacent faces. To find corners, ADSODA iterates through every possible *n*-tuple of faces, and finds their intersection (a point). If the intersection lies on the boundary of the solid, it is a corner. If it lies outside the solid, it is not a corner, and it is ignored. For each corner thus found, every face which contributed to that corner is added to the "adjacent faces" list of every other face. The corners are also used again later, in rendering.

Once the silhouettes have been computed, they are extruded back into *n*-space, parallel to the viewing vector. They are extruded infinitely into *n*-space, forming a tube through *n*-space which intersects the projection space in the shape of the silhouette, and is parallel to the viewing vector (**Figure 5**). This extrusion is easily represented using ADSODA's normal solid object representation-- it is the intersection of the set of halfspaces in *n*-space which intersect the destination space at the edges of the silhouette, and which are perpendicular to the viewing plane. This is the *silhouette tube*.

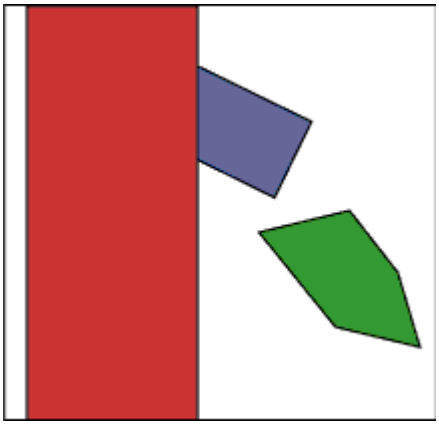


Figure 5a

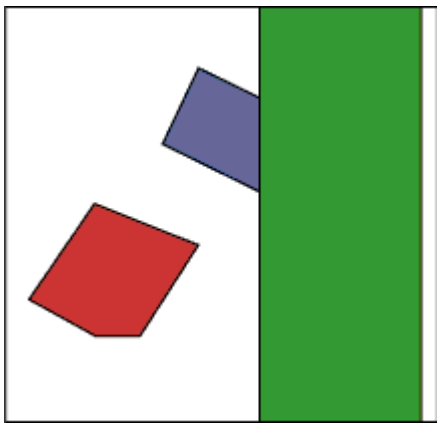


Figure 5b

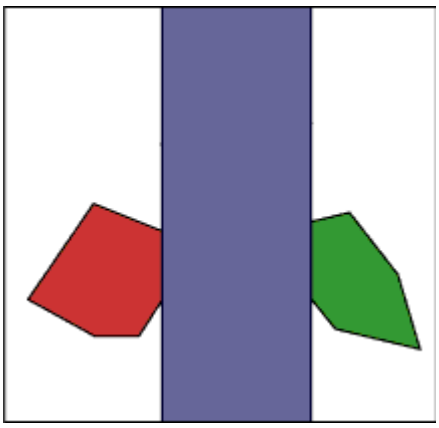


Figure 5c

The silhouette tube is then subtracted from every solid in the space which is behind the silhouetted solid (a solid S1 is behind another S2 if some corner of S1 is inside the silhouette tube of S2, and if that corner is behind a backface of S2), and the rear solid is replaced by the difference. This difference is exactly that portion of the back solid which is not obscured by the silhouetted solid.

To compute the difference of two solids (S1 - S2), ADSODA duplicates S1, and the copy is repeatedly sliced by faces of S2. Each slice results in another piece S1 which is outside the volume of S2. The union of all these pieces is the difference. (Incidentally, the final piece which remains after all the pieces of the difference have been sliced away is the intersection).

For example, **Figure 6** shows the subtraction/intersection process for two two-dimensional solids. Initially, S1 is duplicated, creating the red solid. In **Figure 6b**, this red solid sliced against one of the faces of S2, by adding that face as a face of the red solid. Since this face does not change the appearance of the red solid (the red solid already lies entirely in the halfplane described by that face), this face is discarded automatically by ADSODA, and the process continues. The next face (**Figure 6c**) is more interesting; adding this face to the red solid changes it (cuts off a corner). The new red solid is computed by copying the red solid, and adding the face; the green solid is computed by copying the previous red solid again, and adding the negative of the face (the face with its normal negated; the vectors in the diagrams are the negative normals). The green solid is

added to the difference, and the process continues with the new sliced red solid. In this manner, for each face which intersects the red solid, the red solid is sliced smaller against that face, and the piece which is sliced off is added to the difference list (the list of solids which represents the difference). At the end, the red solid is the intersection, and the green solids are the difference.

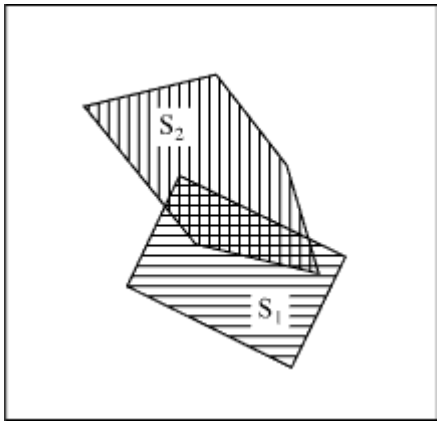


Figure 6a

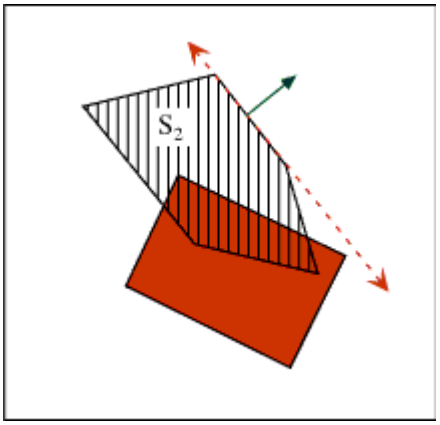


Figure 6b

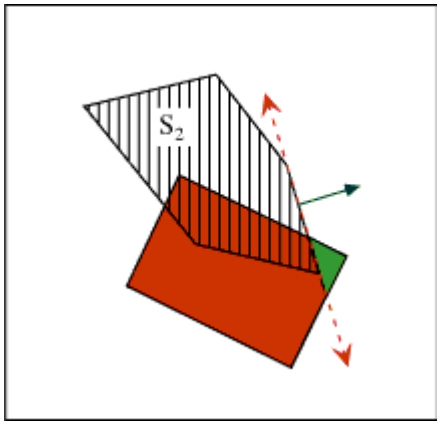


Figure 6c

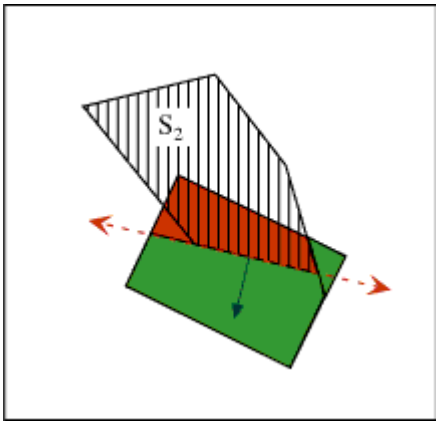


Figure 6d

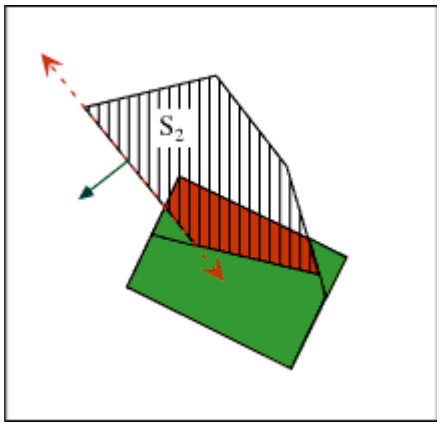


Figure 6e

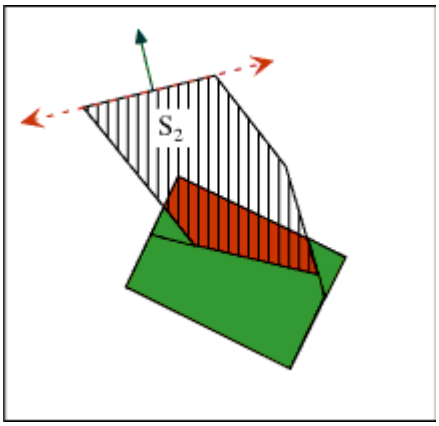


Figure 6f

When the hidden solid removal algorithm is done, all solids in the space will have been sliced by the silhouette tubes of all solids in front of them, and all pieces which lay in the silhouette tubes will have been removed from the space (**Figure 7**). This ensures that everything which was obscured in the original space will be missing in the resulting space, so projection can safely continue; all projected solids will be distinct.

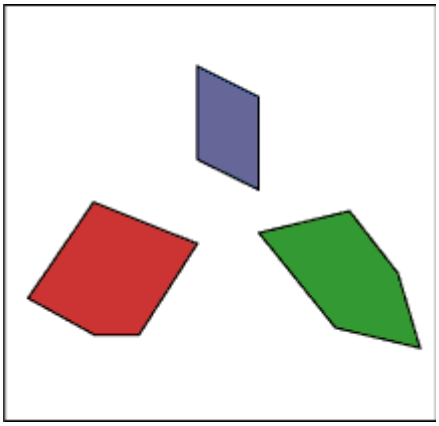


Figure 7

Projection

The projection algorithm works in much the same way as the silhouette algorithm. For each frontface, the edges shared with all adjacent faces (not just backfaces) are computed, and are projected to the destination space (**Figure 8**). The combination of all projected edges defines the projected face as a single solid in the destination space using the same representation as is used elsewhere (intersection of halfspaces). Note that in **Figure 8**, lighting has also been applied to the projection, as though a white light were coming from negative *y*; the faces which face most directly towards negative *y* are the brightest.

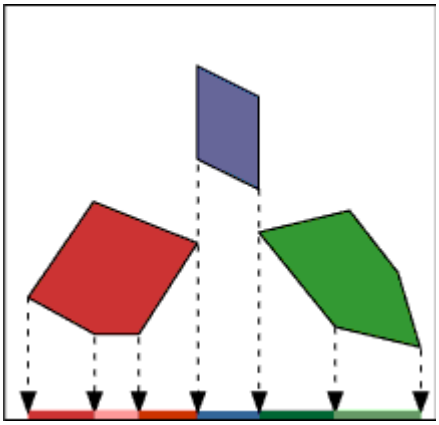


Figure 8

Rendering in 2D

ADSODA uses OpenGL to render solids (polygons) in 2D. It draws the outline of the polygon by starting at the top corner (in screen coordinates), connecting the dots down the right side, and connecting the dots back up the left side. Since there is no information in the basic representation about the order corners should be drawn in, it is necessary to explicitly compute this order. ADSODA does this by sorting corners on their *y* coordinates, finding the top and bottom points (first and last in the sort), using the line segment between them to find which side of the polygon the other corners lie on, and then traversing the corners as described above (down the right, up the left).

Rendering in 3D

ADSODA can use OpenGL to render solids in 3D. It draws each face of the solid as a 3-dimensional polygon. Since there is no information about the order to draw the vertices in, ADSODA must determine this before it draws. It does this by arbitrarily choosing two corners, p_1 and p_2. For each corner from p_2 to p_n, it determines the angle between p_1p_2 and p_1p_n using the following formula:

$$\tan \phi = \frac{\left| p_1p_2 \times p_1p_n \right|}{p_1p_2 \cdot p_1p_n}$$

It negates the resulting angle when the face normal faces away from p_1p_2 x p_2p_n. This gives an ordering which can be used directly to order the drawing of the vertices.

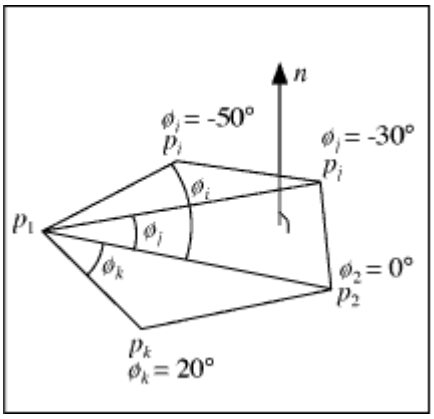


Figure 9

It's also possible to render a three-dimensional space with ADSODA by performing the projection to two dimensions using ADSODA's algorithms, and then rendering it in 2D, but this is slower than taking advantage of OpenGL's fast rendering algorithms.

Implementation Details

ADSODA is written in C++. It requires the Standard Template Library, and uses a very limited portion of OpenGL for rendering (polygons, polylines, and colors). It has been successfully built using both g++, the SGI C++ compiler, and Microsoft Visual C++. It implements the following classes:

- **Vector**: an *n*-dimensional vector
- **AMatrix**: an *n*-by-*n* matrix
- **Color**: a color
- **Face**: one face of a Solid
- **Halfspace**: an *n*-dimensional halfspace
- **Light**: an *n*-dimensional light
- **Solid**: an *n*-dimensional solid object (a collection of Faces)
- **Space**: an *n*-dimensional space (a collection of Solids, and a collection of Lights).

The file demo.cpp implements 2-, 3-, and 4-dimensional demonstrations.

You can download the source code for ADSODA from here.

Credits

Thanks to Antonio Costa, whose implementation of CSG in his rtrace ray tracing program helped to inspire the CSG-style solid representation used in ADSODA.

Thanks to the many users of my Macintosh program HyperCuber, whose requests spurred me to develop a better way of drawing *n*-dimensional objects.

Thanks to Dr. George Francis, whose Math 428 course at The University of Illinois provided a means for me to invest the time necessary to complete this project, and who provided invaluable help in creating the OpenGL demonstration.

Thanks to Stuart Levy, who provided additional help in getting the demonstration up and running, especially in the CAVE at UIUC.

Thanks to Ben Schaeffer, who helped me to develop the algorithm, used for 3D rendering, which orders the vertices by angle.

Sources

Foley, James D.; van Dam, Andries; Feiner, Steven; Hughes, John F.; *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Company, 1987.

Glassner, Andrew S. (Editor), *Graphics Gems*, Academic Press Professional, 1990.

Schneider, Dennis M., *Linear Algebra: A Concrete Introduction*, Macmillan Publishin Company, New York, 1987.

Related Links

Jean-Francois Bigot has written a program called 4DNav, using the Factor programming language, which uses ADSODA as its algorithm for hidden solid removal.

ADSODA was created by Greg Ferrar.