NEW PROGRAMS

Visualization of the relationship between Fischer and Haworth projections of monosaccharides by animation on a microcomputer

J.F. Ewart, G.H. Kirby and J.D. Rayner

Department of Computer Science, University of Hull, UK

An animation is described to assist in the visualization of the relationship between the Fischer and Haworth projections of monosaccharides. The animation helps the understanding of two quite different, commonly used representations of the same molecule. The animation runs on a 386-based PC with VGA graphics. The method for realizing such an animation on this hardware is described in detail; C code is provided in the Appendix.

Keywords: animation, PC graphics, monosaccharides, Fischer projection, Haworth projection

INTRODUCTION

During work to include saccharide names in software for translation of systematic organic chemical nomenclature to structure diagrams,[1] both the Fischer and the Haworth projections were seen as appropriate endpoints. Both are used quite generally in the literature and in education[2] for structure diagrams of saccharides, each emphasizing a different aspect of the three-dimensional (3D) structure in the limitations of two dimensions. The Fischer projection is a representation of the connectivities occurring in a molecule with several chiral centers. The spatial arrangement of the atoms is not conveyed. Initially developed for acyclics, the Fischer projections used here are limited to the cyclic structure of monosaccharides. The Haworth projection, on the other hand, attempts to give an indication of the 3D shape of a saccharide molecule, although in an abstract form without consideration for the precise geometry. The ring is drawn in a plane at 30° to the paper or screen, with atoms and groups attached by vertical bonds. By convention, the oxygen atom is on the far side (away from the viewer) and the ring bonds (nearer the viewer) may be thickened to emphasize the perspective view.

The projections are illustrated in Figure 1, which is a screen dump of the display after translation of the name alpha-D-glucopyranose by the Hull University Nomenclature Translator (HUNT) software.[1] The Fischer projection is on the left, the Haworth projection in the center and a second Haworth projection is on the right that can be manipulated by the user. The user can either flip (menu option) this latter projection by 180° (rotate around the axis perpendicular to the screen) or rotate it anticlockwise (menu option) in 60° increments about the axis perpendicular to the plane of the ring. In Figure 1 it has been flipped and then rotated by 180°.

When both Fischer and Haworth representations are displayed, it can be quite confusing as to which atom in one projection corresponds to the same atom in the other. Many generations of students and chemists have spent time trying to imagine how one representation maps to the other. The reader is encouraged to try this on the Fischer and Haworth projections illustrated in Figure 1. Given that we wished to provide both projections as the output from the translation of a saccharide name, a suitable way of providing this visualization was by animating the transition between Fischer and Haworth projections (the aniMation menu option seen in Figure 1). The objective was to display, on screen, what the users must do in their heads in order to understand the relationship between the projections.

ANIMATION

The animated sequence was written in Borland Turbo C for a 386-based PC with VGA graphics. This is the hardware platform for HUNT. Only the ring skeleton was drawn so that the animation would be as effective and uncluttered as possible for visualization of the relationship between the two

Address reprint requests to Dr. Kirby, Department of Computer Science, University of Hull, Hull HU6 7RX, UK, Tel.: +44 482 465951, Fax: +44 482 466666.
Received 5 October 1993; accepted 30 November 1993

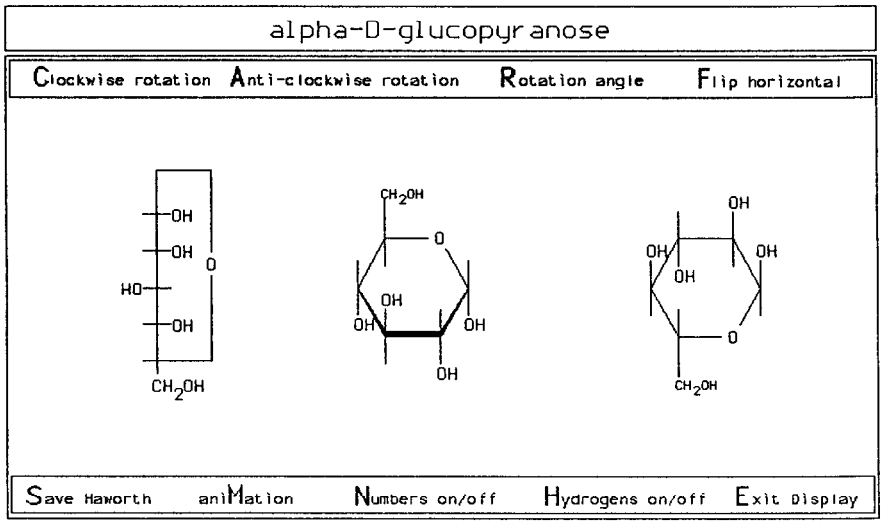

Figure 1. Fischer and Haworth projections displayed by HUNT after translation of the name alpha-D-glucopyranose.

projections. Other substituent groups, which may be displayed after nomenclature translation as shown in Figure 1, were found to complicate the display at the animation stage. The monosaccharide skeletons in the projections are drawn very simply as a collection of straight lines. These can be programed and drawn easily and quickly using the graphics library supplied with the compiler. This proved to be a simplifying factor in implementing the animation.

The Fischer and Haworth projections are merely representations of the structure of the saccharide molecule, so there was no geometric restriction in how one projection was transformed to the other on the screen. The animation sequence was not restricted by any considerations of bond lengths and angles. For the animation sequence, the projections were further simplified to a rectangle for the Fischer, with the CH_2OH group projecting as a line extending the spine of carbons at the bottom, and a pentagon, hexagon or heptagon for the Haworth. These are seen in Figure 2 at the start and finish of the animation sequence.

The keyframe animation method[3] was used whereby the two extreme structures were used to interpolate other frames in order to make the animation smooth. Each frame is characterized by the same number of key points that must correspond and that define the shapes at the initial and final frames for the interpolation procedure. However, there are three nodes in the Fischer diagram that need to be represented, but do not correspond to carbon atoms. These nodes form three of the angles of the rectangle forming the ring closure and have effectively to be lost by being assigned positions on the Haworth ring. It would be possible to locate all the extra nodes on existing carbon or oxygen nodes (any of the six vertices on the hexagon). However, this tends to bias the calculation of intermediate frames, as certain nodes have a heavier weighting than others. The approach adopted is to place the nodes suitably along the bonds on either side of the oxygen atom in the Haworth projection. They can also be seen disappearing into the ring near the oxygen atom in the last three intermediate frames on the right-hand side of Figure 2. This also produces a certain amount of weighting on one side of the Haworth ring, but it is less pronounced than weighting individual atoms.

Having computed all the frames for the animation, with screen coordinates of nodes stored in a suitable data structure, a means of displaying them rapidly in sequence was needed, appropriate to the 386-based PC being used.

The color look-up table technique developed by Shoup,[4] associates a unique logical color with each of the frames, all of which are drawn and displayed at the same time on the screen. For a monochrome display, which is acceptable for this animation, all the logical colors are initially set to background (black). The animation is generated by iterating the logical colors in the sequence of the displayed frames, setting each logical color to white for a short time before resetting it to black. While this method can produce fast, flicker-free animations on quite modest hardware, the number of frames in the animation is limited by the size of the color palette. In our case, we had 16 colors including black and white, which left only 14 colors for association with frames. After some experimentation, this was found to be too small a number of frames for an effective animation.

A second technique considered was the computation and storage of the frames for the animation in video pages of the video adapter. Essentially, a video page is a representation of the screen as a portion of memory. Video pages can be saved and reloaded through calls to graphics library functions in Borland Turbo C.[5] This allows fast animation by loading each of the video pages in turn. However, the method is of very limited application on PCs, as the available memory allows, at most, four video pages to be stored. This was worse than the color look-up table method.

The solution adopted relied on being able to draw an image consisting of about seven short, straight lines, depending on the ring size, almost instantaneously on the screen of a modern PC (typically a 386-based PC running at 25 MHz with VGA graphics). Each stored frame was drawn in sequence directly to the screen in a non-background color and, after a short pause, redrawn in the background color to remove it. The advantages of this method over the two other methods considered were that no limit was imposed on the number of frames in the sequence (32 intermediates were used) and that different colors could be used as part of the animation. This allows color coding of the different atoms or highlighting of other features in the molecule. In our experience, drawing the oxygen atom and color coding the carbon atoms did not noticeably affect the animation speed.

This was felt to give an acceptable animation even without showing the oxygen and carbon atoms in intermediate frames. In Figure 2, these atoms are only included for emphasis and only

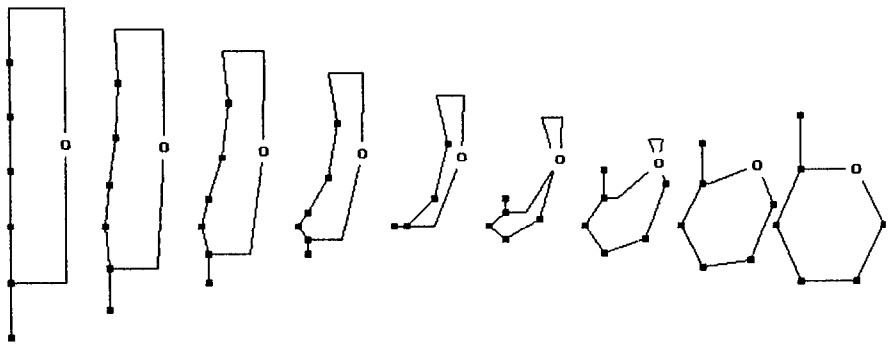

Figure 2. Initial, seven of the intermediate and final frames of the animation of the Fischer-to-Haworth transformation.

every fourth intermediate frame is shown. This gives an impression of the animation without cluttering up the figure with all 32 frames. On the screen, only one of the frames is visible at one time. Furthermore, to help the user with the visualization of the relationship between the projections, the animation is repeated automatically so that on each repetition the user can concentrate on some, possibly different, aspect of the transformation.

The animation speed can be controlled effectively by including a delay between the time taken to draw an image on the screen and the time taken to remove it. In practice, it is simply a case of including an empty loop between the draw and redraw routines, with the number of iterations controlling the delay. This is a very simple method, but it does depend on the clock speed of the hardware. Other more accurately timed delays would be necessary at this point if the software were to be run on a variety of platforms.

The C code developed for this work is outlined in the Appendix. To save space it is restricted to pyranose rings and does not allow variation of the speed of the animation. The extension of the code for other ring sizes, variation in animation speed and direction, and use of color is straightforward and will require minimal input data.

DISCUSSION

Animations involving molecular structures reported in the literature are often, not surprisingly, connected with molecular motion.[6] Such work relies on powerful graphics workstations and large quantities of data, possibly computed on other even more powerful computers. The power of animation for visualization lies not just at the top end of the graphics market. Henkel[7] has drawn attention to the value of animation in the classroom for PC-based modeling. He used animation to demonstrate the interaction of enzymes and receptors with their substrates and ligands, respectively.

The use of animation to visualize the relationship between different representations of the same structure is different and could find other applications. The description here of the ways of realizing the animation may encourage others to simplify their displays to a limited number of lines and hence use the PC for animation. Our work may be of value as part of a computer-assisted learning package on saccharides and the code has been included to encourage such use.

There is no reason why the technique would not cope with oligosaccharides as well as monosaccharides. It is, however, debatable whether the animation of Fischer projection oligosaccharides to Haworth projections would improve the understanding of the two structure diagrams. It should be noted that the depiction of the oligosaccharide structure is predominantly shown in the Haworth form.

The software could be extended to illustrate the corresponding stereochemistry between the two projections if substituents were added to the frames in the animation. However, to achieve a clear representation in the intermediate frames, additional key points would have to be defined for the substituents and some symbolism or color coding adopted to represent the configurations.

REFERENCES

1 Cooke-Fox, D.I., Kirby, G.H., Lord, M.R. and Rayner, J.D. Computer translation of IUPAC systematic organic chemical nomenclature. 4. Concise connection tables to structure diagrams. *J. Chem. Inf. Comput. Sci.* 1990, **30**, 122–127

2 El Khadem, H.S. *Carbohydrate Chemistry. Monosaccharides and their Oligomers;* Academic Press: San Diego, 1988; 32–45

3 Magnenat-Thalmann, N. In "Computer animation: a tool that gives life to visualization," *Scientific Visualization and Graphics Simulation* (D. Thalmann, Ed.) Wiley, New York (1990) 113–128

4 Shoup, R.G. Color table animation. *ACM SIGGRAPH* 1979, **13** (2) 8–12

5 *Turbo C User's Guide, Chapter 8;* Borland International: Scotts Valley, CA, 1988

6 Swanson, S.M., Wesolowski, T., Geller, M. and Meyer, E.F. Animation: A useful tool for protein molecular dynamicists, applied to hydrogen bonds in the active site of elastase. *J. Mol. Graphics* 1989, **7** (4) 240–242

7 Henkel, J.G. PC-based molecular modeling in the classroom—Applications to medicinal chemistry and biochemistry. *J. Mol. Graphics* 1991, **9** (1) 11–17

```
#include <stdio.h>
#include <graphics.h>

#define FRAME 33
#define NOOFPOINTS 12
#define FACTOR 18

struct coord_type {
    int points[NOOFPOINTS] [4];
};

void setup ();
void cleararray(struct coord_type *c);
void getdata(int cc, struct coord_type *c);
void inbetween(int cc, struct coord_type *c);
void differ(struct coord_type *x,struct coord_type *y,struct coord_type *z);
void draw_sugar(int cc, struct coord_type *c);
void undraw_sugar(int cc, struct coord_type *c);
void ring_oxygen(int xx, int yy);
void blob(int xx, int yy);

main() {
    struct coord_type image [FRAME];
    int i, delay, delay2;

    setup();
    cleararray(image);

    getdata(1, image);
    getdata(2, image);

    for(i=0; i<FRAME; i++) {
        inbetween(i+1, image);
    } /* for - calculate coordinates */

    for(i=0; i<FRAME; i+=4) {
        draw_sugar(i, image);
        for (delay=0; delay<1000; delay++)
            for (delay2=0; delay2<100; delay2++) {};
        if (i!=0 && i!=FRAME-1) undraw_sugar(i, image);
    } /* for - display and remove the images */

    getch();
    closegraph(); /* close graphics screen */

    return(0);
} /* main */

void setup() {
    int gdriver = DETECT, gmode;

    initgraph(&gdriver, &gmode, "");
    if (graphresult() != grOk) {
        printf("ERROR\n");
        exit();
    } /* if - error occurred opening graphics device */

    moveto(5, 5);
} /* setup - initialise graphics screen */
```

```
void cleararray(struct coord_type *c) {
    int i, j, k;

    for(i=0;i<FRAME;i++) {
       for(j=0; j<NOOFPOINTS; j++) {
          for(k=0; k<4; k++)
             (*(c+i)).points[j] [k] = 0;
       } /* for - j */
    } /* for - i*/
} /* cleararray - initialise arrays to be used in the following calculations*/

void getdata(int cc, struct coord_type *c) {
    switch(cc) {
        case 1 :
            (*(c+0)).points[0] [0]=20; (*(c+0)).points[0] [1]=360; (*(c+0)).points[0] [2]=6;
            (*(c+0)).points[1] [0]=20; (*(c+0)).points[1] [1]=320; (*(c+0)).points[1] [2]=6;
            (*(c+0)).points[2] [0]=20; (*(c+0)).points[2] ]1]=280; (*(c+0)).points]2] [2]=6;
            (*(c+0)).points[3] [0]=20; (*(c+0)).points[3] [1]=240; (*(c+0)).points[3] [2]=6;
            (*(c+0)).points[4] [0]=20; (*(c+0)).points[4] [1]=200; (*(c+0)).points[4] [2]=6;
            (*(c+0)).points[5] [0]=20; (*(c+0)).points[5] [1]=160; (*(c+0)).points[5] [2]=6;
            (*(c+0)).points[6] [0]=20; (*(c+0)).points[6] [1]=120; (*(c+0)).points[6] [2]=0;
            (*(c+0)).points[7] [0]=60; (*(c+0)).points[7] [1]=120; (*(c+0)).points[7] [2]=0;
            (*(c+0)).points[8] [0]=60; (*(c+0)).points[8] [1]=220; (*(c+0)).points[8] [2]=8;
            (*(c+0)).points[9] [0]=60; (*(c+0)).points[9] [1]=320; (*(c+0)).points[9] [2]=0;
            (*(c+0)).points[10][0]=-1
            break; /* pyran - Fischer projection */

        case 2 :
            (*(c+FRAME-1)).points[0] [0]=20; (*(c+FRAME-1)).points[0] [1]=200; (*(c+FRAME-1)).points[0] [2]=6;
            (*(c+FRAME-1)).points[1] [0]=20; (*(c+FRAME-1)).points[1] [1]=240; (*(c+FRAME-1)).points[1] [2]=6;
            (*(c+FRAME-1)).points[2] [0]=20; (*(c+FRAME-1)).points[2] [1]=280; (*(c+FRAME-1)).points[2] [2]=6;
            (*(c+FRAME-1)).points[3] [0]=20; (*(c+FRAME-1)).points[3] [1]=320; (*(c+FRAME-1)).points[3] [2]=6;
            (*(c+FRAME-1)).points[4] [0]=60; (*(c+FRAME-1)).points[4] [1]=320; (*(c+FRAME-1)).points[4] [2]=6;
            (*(c+FRAME-1)).points[5] [0]=80; (*(c+FRAME-1)).points[5] [1]=280; (*(c+FRAME-1)).points[5] [2]=6;
            (*(c+FRAME-1)).points[6] [0]=65; (*(c+FRAME-1)).points[6] [1]=250; (*(c+FRAME-1)).points[6] [2]=0;
            (*(c+FRAME-1)).points[7] [0]=65; (*(c+FRAME-1)).points[7] [1]=250; (*(c+FRAME-1)).points[7] [2]=0;
            (*(c+FRAME-1)).points[8] [0]=60; (*(c+FRAME-1)).points[8] [1]=240; (*(c+FRAME-1)).points[8] [2]=8;
            (*(c+FRAME-1)).points[9] [0]=20; (*(c+FRAME-1)).points[9] [1]=240; (*(c+FRAME-1)).points[9] [2]=0;
            (*(c+FRAME-1)).points[10] [0]=-1
            break; /* pyran - Haworth projection */

        /* note - data for septan and furan rings can be included here as well */
    } /* switch - offsets for the Fischer and Haworth projections */
} /* getdata - get data for Haworth and Fischer rings*/

void inbetween(int cc, struct coord_type *c) {
    switch(cc) {
        case 1 : differ((c+0),(c+FRAME-1),(c+16)); break;
        case 2 : differ((c+0),(c+16),(c+8)); break;
        case 3 : differ((c+16),(c+FRAME-1),(c+24)); break;
        case 4 : differ((c+0),(c+8),(c+4)); break;
        case 5 : differ((c+8),(c+16),(c+12)); break;
        case 6 : differ((c+16),(c+24),(c+20)); break;
        case 7 : differ((c+24),(c+FRAME-1),(c+28)); break;
        case 8 : differ((c+0),(c+4),(c+2)); break;
        case 9 : differ((c+4),(c+8),(c+6)); break;
        case 10 : differ((c+8),(c+12),(c+10)); break;
        case 11 : differ((c+12),(c+16),(c+14)); break;
        case 12 : differ((c+16),(c+20),(c+18)); break;
        case 13 : differ((c+20),(c+24),(c+22)); break;
        case 14 : differ((c+24),(c+28),(c+26)); break;
        case 15 : differ((c+28),(c+FRAME-1),(c+30)); break;
```

```
        case 16 : differ((c+0),(c+2),(c+1)); break;
        case 17 : differ((c+2),(c+4),(c+3)); break;
        case 18 : differ((c+4),(c+6),(c+5)); break;
        case 19 : differ((c+6),(c+8),(c+7)); break;
        case 20 : differ((c+8),(c+10),(c+9)); break;
        case 21 : differ((c+10),(c+12),(c+11)); break;
        case 22 : differ((c+12),(c+14),(c+13)); break;
        case 23 : differ((c+14),(c+16),(c+15)); break;
        case 24 : differ((c+16),(c+18),(c+17)); break;
        case 25 : differ((c+18),(c+20),(c+19)); break;
        case 26 : differ((c+20),(c+22),(c+21)); break;
        case 27 : differ((c+22),(c+24),(c+23)); break;
        case 28 : differ((c+24),(c+26),(c+25)); break;
        case 29 : differ((c+26),(c+28),(c+27)); break;
        case 30 : differ((c+28),(c+30),(c+29)); break;
        case 31 : differ((c+30),(c+FRAME-1),(c+31)); break;
    } /* switch */
} /* inbetween - evaluate images between Haworth and Fischer projections */

void differ(struct coord_type *x,struct coord_type *y,struct coord_type *z) {
    int i;

    for(i=0<NOOFPOINTS && (*x).points[i][0]>0;i++) {
        (*z).points[i] [0] = (int) ((*x).points[i] [0]+(*y).points[i] [0])/2;
        (*z).points[i] [1] = (int) ((*x).points[i] [1]+(*y).points[i] [1])/2;
        (*z).points[i] [2] = (*x).points[i] [2];
    } /* for - i */
} /* differ - calculate the mean of two matrices */

void draw_sugar(int image_num, struct coord_type *c) {
    int count, x_inc, x_coord, y_coord, oxygen_x, oxygen_y;

    setcolor(15); /* set pen to solid, white line */
    x_inc = (image_num * FACTOR) - FACTOR;
    moveto((*(c+image_num)).points[0] [0]+x_inc, (*(c+image_num)).points[0] [1]);
    for(count=1; ((count<NOOFPOINTS-1) && ((*(c+image_num)).points[count][0]>0)); count++) {
        x_coord = (* (c+image_num)).points[count] [0]+x_inc;
        y_coord = (* (c+image_num)).points[count] [1];
        lineto(x_coord, y_coord);
        if ((image_num==0 || image_num==FRAME-1) && ((*(c+image_num)).points[count-1] [2] == 8)) {
            ring_oxygen((*(c+image_num)).points[count-1] [0]+x_inc, (* (c+image_num)).points[count-1] [1]);
        } /* if - show ring oxygen only in first and last images */
        if (((*(c+image_num)).points[count-1] [2] == 6)) {
          blob((*(c+image_num)).points[count-1] [0]+x_inc, (* (c+image_num)).points[count-1] [1]);
        } /* if - show carbons only in first and last images */
        moveto(x_coord, y_coord);
    } /* for - draw ring*/
    lineto((*(c + image_num)).points[1] [0]+x_inc, (* (c+image_num)).points[1] [1]);
} /* draw_sugar - draw Haworth, Fischer and intermediate images */

void undraw_sugar(int cc, struct coord_type *c) {
    int i,x_inc;

    setcolor(0);
    x_inc = cc*17;
    moveto((*(c+cc)).points[0] [0]+x_inc, (*(c+cc)).points[0] [1]);
    for(i=1; ((i<NOOFPOINTS-1) && ((*(c+cc)).points[i][0]>0)); i++) {
       lineto((*(c+cc)).points[i][0]+x_inc, (*(c+cc).points[i] [1]);
    } /* for */
    lineto((*(c+cc)).points[1] [0]+x_inc, (*(c+cc)).points[1] [1]);
} /* undraw_sugar - remove previously drawn image */
```

```
void ring_oxygen(int xx, int yy) {
    int i,xi,yi;

    setcolor(0);
    xi = xx-8; yi = yy-8;
    for(i = 1;i<16;i++) {
        moveto(xi + i,yi);
        lineto(xi + i,yi + 16);
    } /* for - background colour */
    setcolor(15);
    moveto(xx-3,yy-3);
    outtext ("O");
} /* ring_oxygen - draw oxygen atom on ring */

void blob(int xx, int yy) {
    int count, size = 2;

    for (count = 0; count<= size; count++) {
        moveto(xx-size, yy-count);
        lineto(xx + size, yy-count);
    } /* for draw a succession of horizontal lines */
    for (count = 1; count<= size; count++) {
        moveto(xx-size, yy + count);
        lineto(xx + size, yy + count);
    } /* for draw a succession of horizontal lines */
} /* blob - draw a filled rectangle for each carbon atom */
```