You will write a program to animate the slider-crank linkage that you designed for your juice extractor in Project #1. The program will use the “graphics pipeline” to draw each link of the juice extractor at each position used in the animation. The animation will be limited to two dimensions for simplicity; i.e., only the plane of motion of the links should be drawn.

You may have used the “mechanism” package of Pro/ENGINEER to animate your linkage in Project #1. In contrast, you will implement the kinematic analysis of the slider-crank yourself in this project. Furthermore, you will implement the details of drawing each link. This approach is taken to prepare you to develop graphics interfaces for other programs in the future. In addition, robotics and advanced kinematics leverage essentially the same background.

You are not expected to utilize the Pro/ENGINEER model of each link. You are not expected to emulate the three-dimensionality of any actual link. You may use simplifications of the actual links; for example, you need not represent fasteners or non-essential details. Nevertheless, you are encouraged to make the 2-D representation of each link as realistic as possible; the overall quality of representation of your mechanism will be considered in the grading.

This document begins with the derivation of the kinematics of a general slider-crank. The general expectations for implementing the graphics animation are then described. A detailed series of implementation notes are provided next to help you develop a well-structured program to implement the animation. A grading sheet is included as the last page.

Students using “C” will use the OpenGL software to implement the animation. Students using FORTRAN will use SGI’s “GL” software. However, the generic transformation and windowing principles to be used in this project are applicable to a wide range of alternative graphics systems.

Kinematics of the Slider-Crank Linkage

The equations necessary for analyzing a generalized slider-crank are developed here. Your animation program will need a function to implement these equations. The results are used to determine the rotations and displacements necessary to orient each link of the slider-crank at each position of the animation.

The generalized slider-crank is illustrated in Fig. 1. We assume that the link pinned to ground is utilized as the driver. The length of the driving link is $r_D$. The length of the

1You may change the kinematics and appearance of your linkage if you wish.
coupler link is $r_C$. The angle of the driver, $\theta_D$, is assumed to be specified. The angle of the path of the sliding link, $\theta_S$, is known. In general, the path of the slider may be offset from the ground pivot of the driver by $f$. At the start of the analysis, the angle of the coupler link, $\theta_C$, and the position of the sliding pivot, $S$, are unknown.

Note that the ground pivot of the driver is assumed to be at the origin of the global coordinate frame in Fig. 1. While this assumption is convenient, it is not necessary. If you prefer not to make this assumption, you will need to add one more translation to your modeling transforms to account for the arbitrary position of the ground pivot.

As with a four-bar linkage, a slider-crank can always be assembled in two different positions for one position of the driver. They are differentiated by assigning a “configuration” when calculating the angle of the coupler link. This is clarified in Fig. 2.

The dashed configuration is obtained by adding angle $\phi_1$ to angle $\theta_S + \pi/2$. This is identified as configuration “$\mu = +1$”. The dotted configuration is obtained by subtracting angle $\phi_1$ to angle $\theta_S + \pi/2$. This is identified as configuration “$\mu = -1$”.

A practical slider-crank may operate in either configuration. You must choose the configuration that you prefer. Note that a practical slider-crank will never change configuration during its motion. The reason is that this would require the coupler to go perpendicular to the slider (angle $\phi_2 = \pm \pi/2$ in Fig. 2). Practically, angle $\phi_2$ should never go out of the range $\pm 30^\circ$, or the slider will likely bind.

Once a configuration is selected, coupler angle $\theta_C$ is calculated by first determining angle $\phi_1$ (see Fig. 3):

$$\phi_1 = \arccos \left( \frac{f - r_D \sin(\theta_D - \theta_S)}{r_C} \right)$$

Then:

$$\theta_C = \theta_S + \frac{\pi}{2} + \mu \cdot \phi_1$$
Figure 2: Two possible configurations of a slider-crank (dashed: $\mu = +1$; dotted: $\mu = -1$).

Figure 3: Detail of the analysis of a slider-crank.
To complete the analysis, the coordinates of the moving pivot of the driving link are calculated, then the coordinates of the slider pin are calculated:

\[ x_A = r_D \cos \theta_D \]
\[ y_A = r_D \sin \theta_D \]
\[ x_S = x_A + r_C \cos \theta_C \]
\[ y_S = y_A + r_C \sin \theta_C \]

**Expectations for Implementing the Graphics Animation**

The graphics pipeline greatly simplifies the process of animating links of arbitrary complexity. The geometry of each link is defined with reference to a convenient local coordinate frame. Modeling transformations are then applied to move the link to its actual position in the mechanism. This procedure is clarified below.

Excellent structure is obtained by designing one function to draw each link of the mechanism. The links are drawn by calling functions similar to “pntMod” in Part 1. You may want to write additional drawing functions to draw line segments, filled polygons, arcs or circles, and the like. For example, we suggest you write a function to draw a line segment named “linMod”. Each drawing function (pntMod, linMod and the like) should accept modeling coordinates to define each entity, along with modeling, viewing and device transforms to map the entity to actual pixel coordinates.

Do not be concerned with the actual \((x, y)\) position and angle of each link when you are defining its geometry. Rather, reference the geometry of each link to a convenient local coordinate frame. For example, if your driving link took the form of a right-angle bell-crank, you might define its geometry as suggested in Fig. 4. Note that point “O” is aligned with the local origin and “O” and “A” are aligned with the local x-axis. The modeling coordinates for the link are then read directly from the local coordinate frame.

When this link is actually drawn, a modeling transform is applied to rotate it by \(\theta_D\). Similarly, the coupler link and the slider link can be rotated and translated to their kinematically correct positions. The procedure is clarified below:

---

You may wish to use “pntMod” directly to draw pivot symbols.
• Position the driving link.
  – Rotate the driver by $\theta_D$ about $O$.

• Position the coupler link.
  – Rotate the coupler by $\theta_C$.
  – Translate the coupler by $(x_A, y_A)$.

• Position the slider link.
  – Rotate the slider by $\theta_S$.
  – Translate the slider by $(x_S, y_S)$.

**Implementation Notes**

1. *Important!* The functions defining each link of the slider-crank *must* define them in *modeling space*! Note that working in modeling space makes defining the geometry the links easier for you. Let graphics transforms do the work of moving the links to their actual positions! If you are attempting to draw the coupler by directly drawing a line between points “A” and “S” in world space, *you are doing it wrong*!

2. This program is complex enough that you need to develop it using structured programming. While we will not collect your structured programming materials, we will examine the structure of your final program. Determine which functions you will need using a top-down design chart. Define the calling arguments of each using an input/output chart. If a function has any logical complexity, simplify it using pseudo-code.

3. Your program will need a loop to draw the slider-crank at every position that you want to show in the animation. Define the viewing and device transforms before starting the loop. Perform the kinematic analysis for a given position of the driver as the first step of this loop. Build the modeling transforms from the results of this analysis. Send the appropriate modeling transform, and the viewing and device transforms, as inputs to the functions that define the geometry of each link. Clearly, a top-down design chart and input-output charts are in order to sort all this out!

4. The linkage will be animated using “double buffering”. Simple programs demonstrating double buffering are available. An OpenGL sample animation is available from file `tchase/openGLDbiBfDemo.c`. A GL sample animation is available from file `tchase/glDbiBfDemo.f`. You are welcome to make use of any fragments of those programs that you wish.

5. You are encouraged to use OpenGL or GL commands in addition to those used for Part 1. OpenGL functions are documented in the *OpenGL Programming Guide (Second Edition)*.

---

3The demonstration programs are also available on the course web site.
Edition) and the OpenGL Reference Manual (Second Edition)\textsuperscript{4}. “GL” functions are documented in the Graphics Library Programming Guide. These manuals are available for check-out both in Room 308 and the Reserve Room of the Engineering Library in Norris Hall.

6. You should maintain viewing and device transformation matrices separate from the modeling transformation matrices. Your program should interactively prompt the user to type in the window and viewport limits. The viewing transform should be defined using these limits. The device transform should map the virtual space to a 600 X 600 pixel window on the workstation screen. The OpenGL or “GL” software will automatically clip at the viewport boundaries using the command sequences described in the 11/10/00 lecture, so you need not implement your own clipping algorithm.

You may combine the viewing and device transformation matrices, or maintain them as two separate matrices, as you prefer.

7. Your program should interactively prompt the user for how many positions of the linkage should be displayed in the animation. If your linkage has a limited range of motion (i.e., the driver is not intended to be rotated in a full circle), you should distribute those positions over the intended range of motion.

8. I suggest that you begin by drawing a very crude “stick figure” representation of each link, using nothing but a function like \texttt{linMod}, as described in the previous section. Successfully implementing nothing but a stick figure animation using the graphics pipeline will qualify you for substantial partial credit for the project.

9. However, you are encouraged to develop more elegant representations of each link. You can add the ability to draw arcs, filled polygons, or other graphic entities by designing functions similar to \texttt{linMod} and \texttt{pntMod} for each graphic primitive that you implement.

10. You should include a representation of the ground link. At a minimum, draw a ground pivot at the base of link 1 and a ground symbol next to the slider (see Fig. 1).

11. You must turn in the following as a grading package:

   (a) The grading sheet, attached as a cover page.
   (b) A 1/2-page overview describing the functionality of your program.
   (c) A 1/2-page instruction sheet on how to operate your program. The instruction sheet should tell how the source code is compiled and run. Moreover, the instructions should define any inputs requested from the user when the program is run. (This will be used by the grader to operate your program.)
   (d) A well-structured, fully-documented program listing. Please do not print more than two pages per 8.5 X 11 sheet on the listing submitted for grading to enable the graders to read it easily.

\textsuperscript{4}“GLUT” function documentation is available from web site http://reality.sgi.com/opengl/glut3/glut3.html.
(e) An electronic version of the source code of your program. The electronic version should consist of all the (uncompiled) “C”, “C++” or FORTRAN code associated with your final program packaged in one file. The file is submitted as follows:

i. Choose a file name that begins with your initials and ends with “.proj3.<ext>.
   (<ext> may be “c”, “cpp” or “f”). The initials will be used to identify your file. For example, I would name my file:
   
   trc proj3.c

ii. Be sure your file is in final form before proceeding with the next step. Due to the nature of our submission procedure, you can only submit one file!

iii. Issue the “ftp” command:
   
   ftp
   You will then get the “ftp>” command prompt. Request a connection to the mechanical engineering ftp server:
   
   ftp> open ftp.me.umn.edu
   Specify your “Name” as “me5241”. Specify your “Password” as “CAE_class”.

iv. Submit your file to the ftp server using the “put” command. For example, I would submit my part file using the command:
   
   ftp> put trc proj3.c

v. Stop ftp by issuing the “exit” or “quit” command:
   
   ftp> quit

Please send your final program to the class ftp site before 8:00 AM on Monday, 11/20/00. You need not turn in any graphics hardcopy, as we will actually run your program to test it.

---

5You may prefer to keep different functions in different files on your own account.

6Characters printed by the system are underlined.
<table>
<thead>
<tr>
<th>Category</th>
<th>Points Possible</th>
<th>Points Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Part 1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>2) Final form of transformation functions</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3) Basic graphics animation</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>4) Representation of link geometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A) Acceptable enhancement</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>B) Exceptional enhancement</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>5) Program structure</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6) Program documentation</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7) Overview and instruction sheet</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>

*Note: Please this sheet as the cover page of your project.*