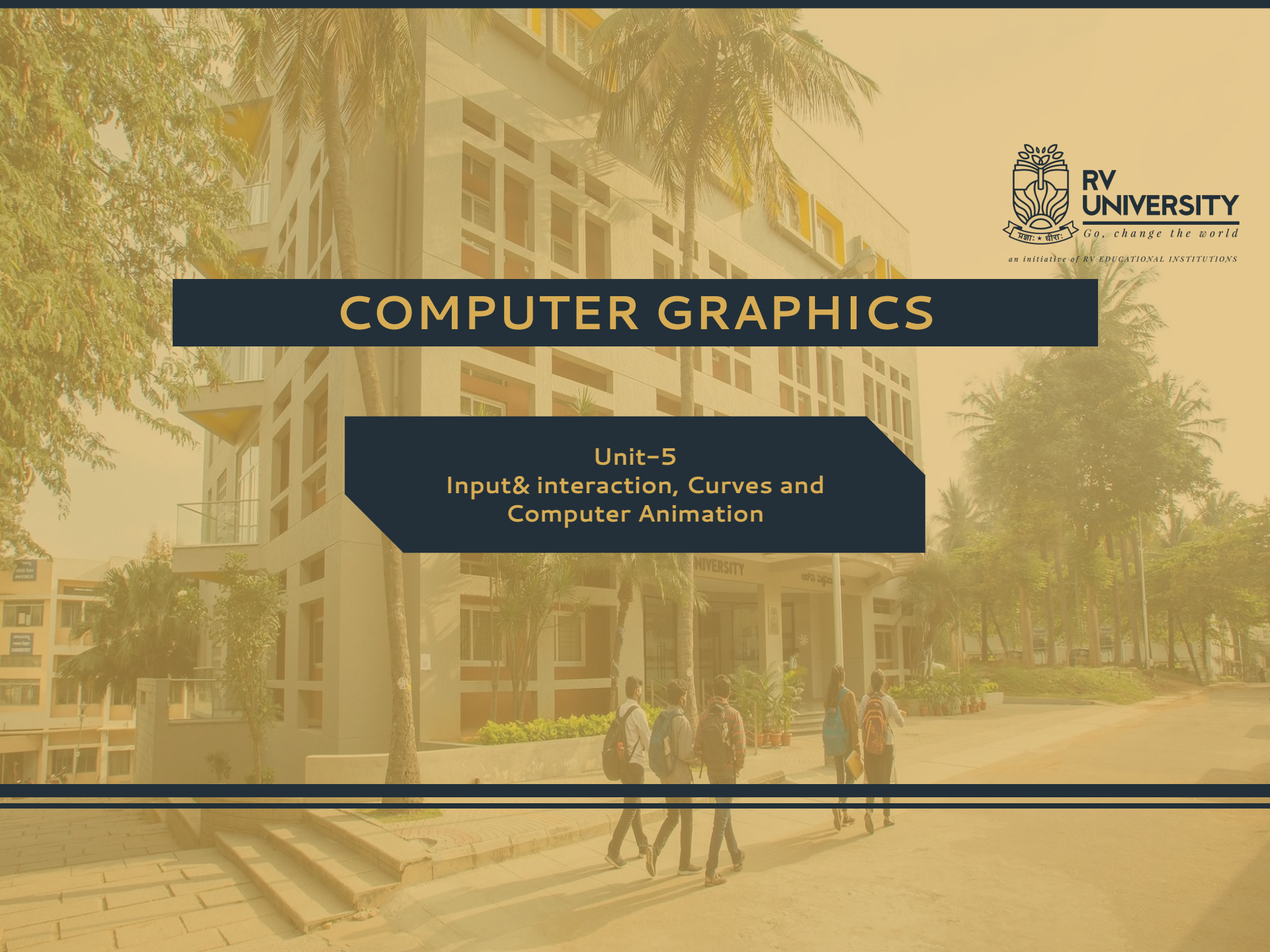


RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

COMPUTER GRAPHICS

Unit–5
Input& interaction, Curves and Computer Animation

Curved surfaces

Equations for objects with curved boundaries can be expressed in either a parametric or nonparametric form.

Quadric surfaces

Superquadrics

Polynomial and exponential functions

Spline surfaces

QUADRIC SURFACES

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

A frequently used class of objects are the quadric surfaces, which are described with second-degree equations (quadratics). They include spheres, ellipsoids, torus, paraboloids and hyperboloids.

More complex objects can be constructed by these objects.

OpenGL Quadric surface and cubic surface functions

- Wireframe surface
- Solid

GLUT functions

Sphere, Cone, Torus, Teapot

GLU functions

- Cylinder
- Tapered cylinder
- Cone
- Flat circular ring(hollow disk)
- Section of a circular ring(disk)

GLUT Quadric-Surface Functions

Sphere

glutWireSphere (r, nLongitudes, nLatitudes);
glutSolidSphere (r, nLongitudes, nLatitudes);

nLongitudes and nLatitudes

select the integer number of longitude and latitude lines that will be used to approximate the spherical surface as a **quadrilateral mesh**

Cone

glutWireCone (rBase, height, nLongitudes, nLatitudes);

glutSolidCone (rBase, height, nLongitudes, nLatitudes);

Torus

glutWireTorus (rCrossSection, rAxial, nConcentrics, nRadialSlices);

glutSolidTorus (rCrossSection, rAxial, nConcentrics, nRadialSlices);

GLUT Cubic surface Teapot Function

glutWireTeapot (size);

or

glutSolidTeapot (size);

- Parameter size sets the double-precision floating-point value for the maximum radius of the teapot bowl.

Figure 13-7 Display of a GLUT sphere, GLUT cone, and GLU cylinder, positioned within a display window by procedure wireQuadSurfs.

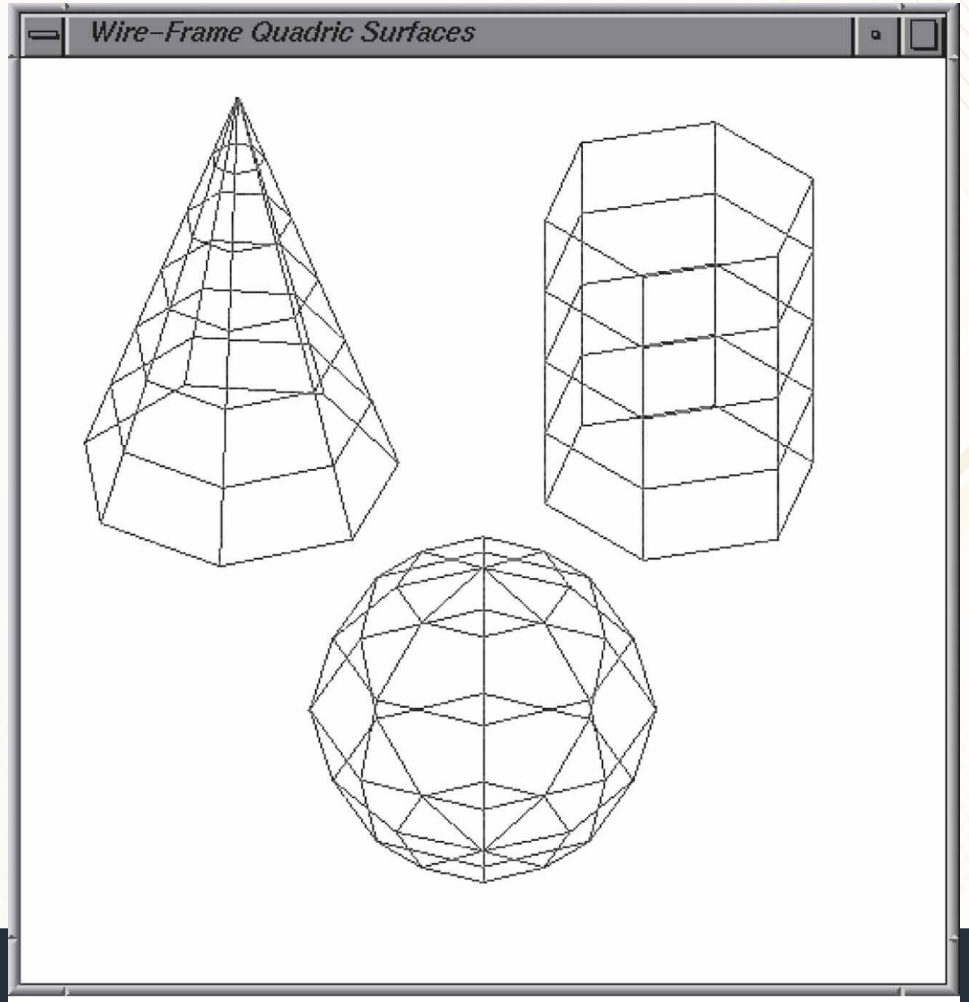


TABLE 13-1

Summary of OpenGL Polyhedron Functions

Function	Description
`glutWireTetrahedron`	Displays a wire-frame tetrahedron.
`glutSolidTetrahedron`	Displays a surface-shaded tetrahedron.
`glutWireCube`	Displays a wire-frame cube.
`glutSolidCube`	Displays a surface-shaded cube.
`glutWireOctahedron`	Displays a wire-frame octahedron.
`glutSolidOctahedron`	Displays a surface-shaded octahedron.
`glutWireDodecahedron`	Displays a wire-frame dodecahedron.
`glutSolidDodecahedron`	Displays a surface-shaded dodecahedron.
`glutWireIcosahedron`	Displays a wire-frame icosahedron.
`glutSolidIcosahedron`	Displays a surface-shaded icosahedron.

TABLE 13-2

Summary of OpenGL Quadric-Surface and Cubic-Surface Functions

Function	Description
`glutWireSphere`	Displays a wire-frame GLUT sphere.
`glutSolidSphere`	Displays a surface-shaded GLUT sphere.
`glutWireCone`	Displays a wire-frame GLUT cone.
`glutSolidCone`	Displays a surface-shaded GLUT cone.
`glutWireTorus`	Displays a wire-frame GLUT torus with a circular cross-section.
`glutSolidTorus`	Displays a surface-shaded, circular cross-section GLUT torus.
`glutWireTeapot`	Displays a wire-frame GLUT teapot.
`glutSolidTeapot`	Displays a surface-shaded GLUT teapot.
`gluNewQuadric`	Activates the GLU quadric renderer for an object name that has been defined with the declaration: `GLUquadricObj *nameOfObject;`

Table 13-2 (continued) Summary of OpenGL Quadric-Surface and Cubic-Surface Functions

`gluQuadricDrawStyle`	Selects a display mode for a predefined GLU object name.
`gluSphere`	Displays a GLU sphere.
`gluCylinder`	Displays a GLU cone, cylinder, or tapered cylinder.
`gluDisk`	Displays a GLU flat, circular ring or solid disk.
`gluPartialDisk`	Displays a section of a GLU flat, circular ring or solid disk.
`gluDeleteQuadric`	Eliminates a GLU quadric object.
`gluQuadricOrientation`	Defines inside and outside orientations for a GLU quadric object.
`gluQuadricNormals`	Specifies how surface-normal vectors should be generated for a GLU quadric object.
`gluQuadricCallback`	Specifies a callback error function for a GLU quadric object.

Bézier Spline Curves

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Spline

- Flexible strip used to produce a smooth curve through a designated set of points
- Curve with a piecewise cubic polynomial functions whose first and second derivatives are continuous across the various curve sections.
- A **piecewise function** is a function in which more than one formula is used to define the output over different pieces of the domain.

Typical computer-aided design (CAD) applications for splines include the design of automobile bodies, aircraft and spacecraft surfaces, ship hulls, and home appliances.

Control points

- We specify a spline curve by giving a set of coordinate positions, called control points, which indicate the general shape of the curve.
- Interpolate splines
- Approximate splines

Interpolation And approximation

Bezier curves are **approximating**. The curve does not (necessarily) pass through all the control points. Each point pulls the curve toward it, but other points are pulling as well.

We'd like to have a spline that is **interpolating**, that is, it always passes through every control point.

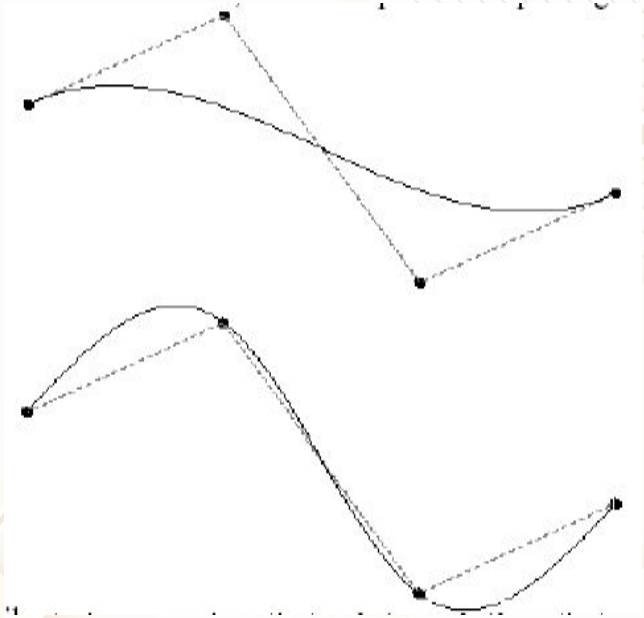


Figure 14-1 A set of six control points interpolated with piecewise continuous polynomial sections.

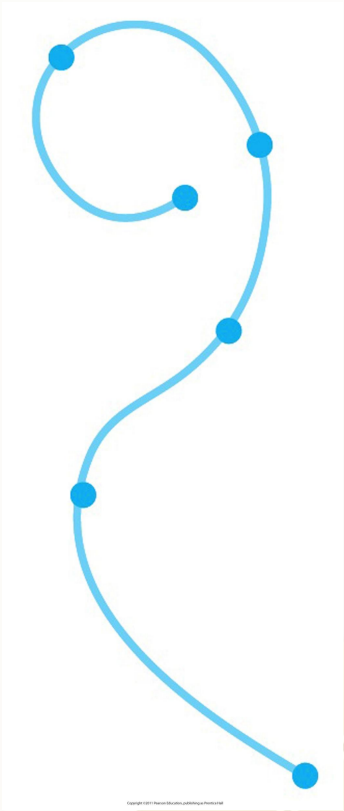
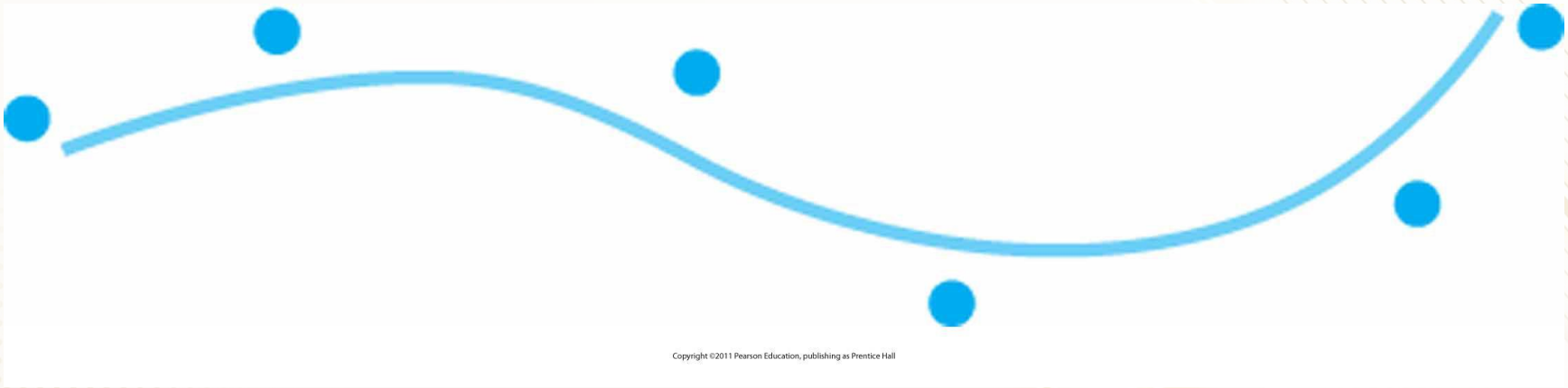


Figure 14-2 A set of six control points approximated with piecewise continuous polynomial sections.



Bézier Spline Curves

This spline approximation method was developed by the French engineer Pierre Bézier for use in the design of Renault automobile bodies.

CAD systems

Can be fitted into any number of control points.

The degree of the Bézier polynomial is determined by the number of control points to be approximated and their relative position

Bézier Spline Curves

Parametric curve that is used to draw smooth lines.

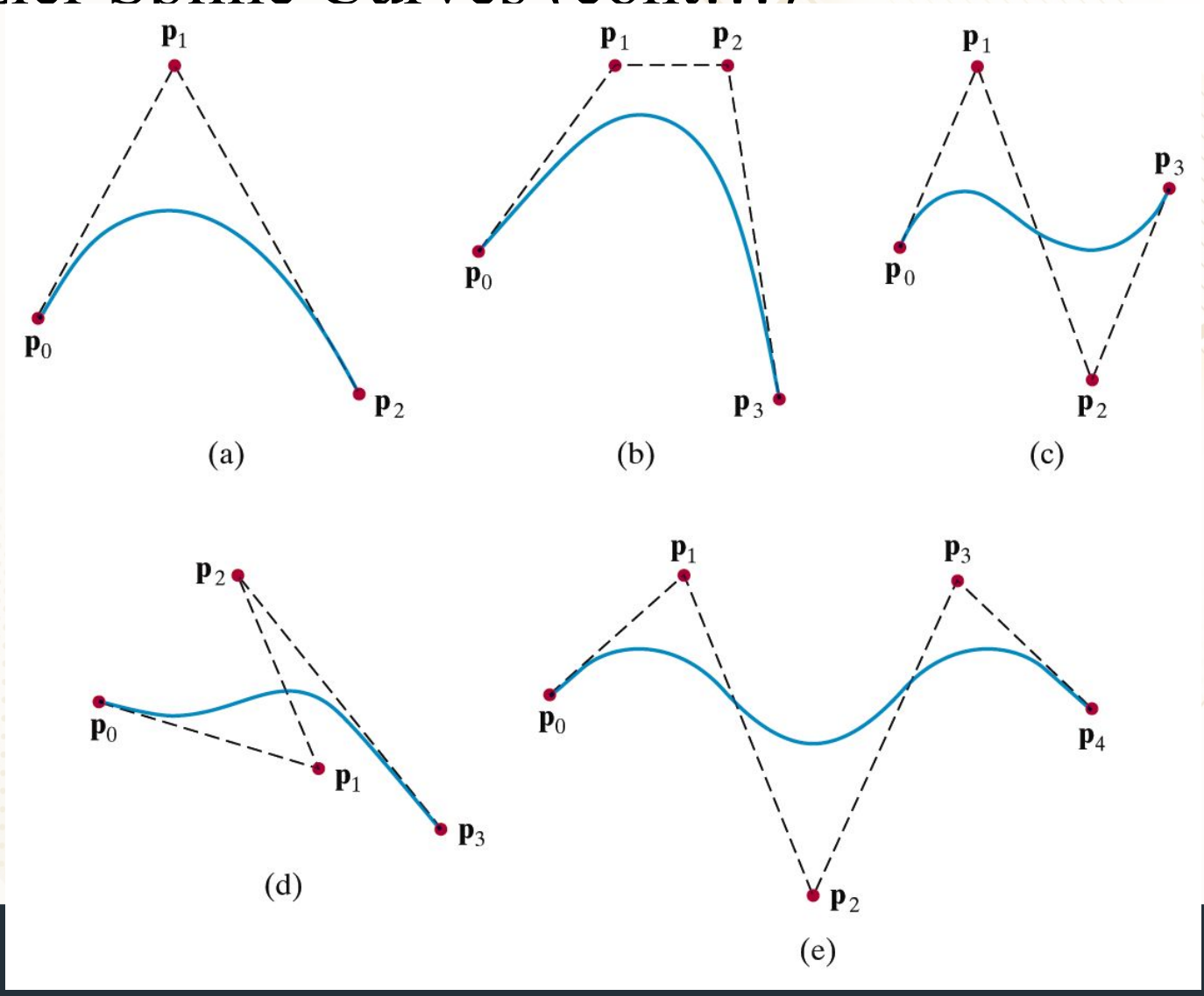
Approximate method

An n degree Bezier curve is defined using n +1 control points

Quadratic curve is defined by 3 control points

Cubic curve is defined by 4 control points

Bézier Spline Curves (cont…)



Bezier Curve equation

We first consider the general case of n+ 1 control-point positions, denoted as $p_k = (x_k, y_k, z_k)$, with k varying from 0 to n.

These coordinate points are blended to produce the following position vector P(u), which describes the path of an approximating Bézier polynomial function between p0 and pn:

$$\mathbf{P}(u) = \sum_{k=0}^{n} \mathbf{p}_k \, \mathrm{BEZ}_{k,n}(u), \qquad 0 \le u \le 1$$

The Bézier blending functions $BEZ_{k,n}(u)$ are the **Bernstein polynomials**

where parameters C(n, k) are the binomial coefficients

$$BEZ_{k,n}(u) = C(n,k)u^k(1-u)^{n-k}$$

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

A set of three parametric equations for the individual curve coordinates:

$$x(u) = \sum_{k=0}^{n} x_k \, \mathrm{BEZ}_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} y_k \, \mathrm{BEZ}_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k \, \mathrm{BEZ}_{k,n}(u)$$

A Bézier curve generated with three collinear control points is a straight-line segment.

A set of control points that are all at the same coordinate position produce a Bézier "curve" that is a single point.

Recursive calculations can be used to obtain successive binomial-coefficient values as

$$C(n,k) = \frac{n-k+1}{k} C(n,k-1)$$

for $n \geq k$. Also, the Bézier blending functions satisfy the recursive relationship

$$\text{BEZ}_{k,n}(u) = (1-u)\text{BEZ}_{k,n-1}(u) + u\,\text{BEZ}_{k-1,n-1}(u), \qquad n > k \geq 1$$

with $\text{BEZ}_{k,k} = u^k$ and $\text{BEZ}_{0,k} = (1-u)^k$.

Properties of Bezier Curves

A useful property is that the curve connects to first and last control points: $\mathbf{P}(0) = \mathbf{p}_0, \quad \mathbf{P}(1) = \mathbf{p}_n$.

The parametric derivative at the end control points is given by:

$$\mathbf{P}'(0) = -n\mathbf{p}_0 + n\mathbf{p}_1, \quad \mathbf{P}'(1) = -n\mathbf{p}_{n-1} + n\mathbf{p}_n.$$

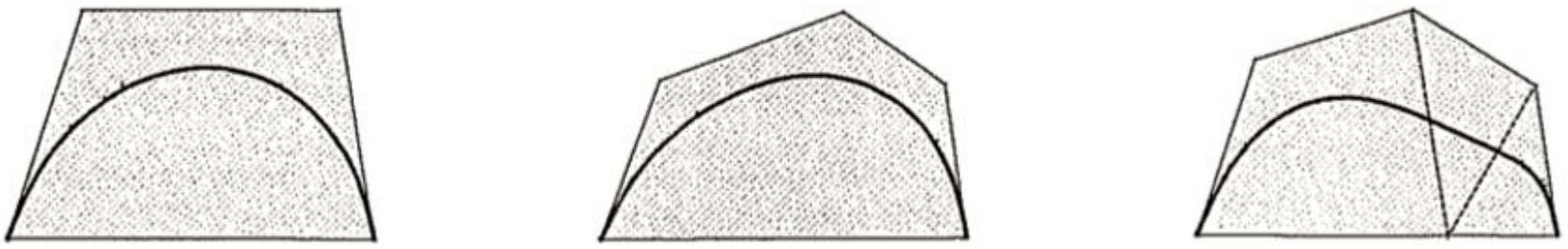
The slope of the cure at the end points is along the line joining the last two end points.

Bezier blending functions are all positive and the sum is always 1.

$$\sum_{i=0}^{m} Bez_{m,i}(u) = 1$$

This means that the curve is the weighted sum of the control points.
Bezier curve lies within the convex hull(convex polygon boundary) of the control points

Convex hull of a Bezier curve is a convex polygon formed by connecting the convex polygon of the curve.



Design Technique using Bezier curves

A closed Bézier curve generated by specifying the first and last control points at the same location.

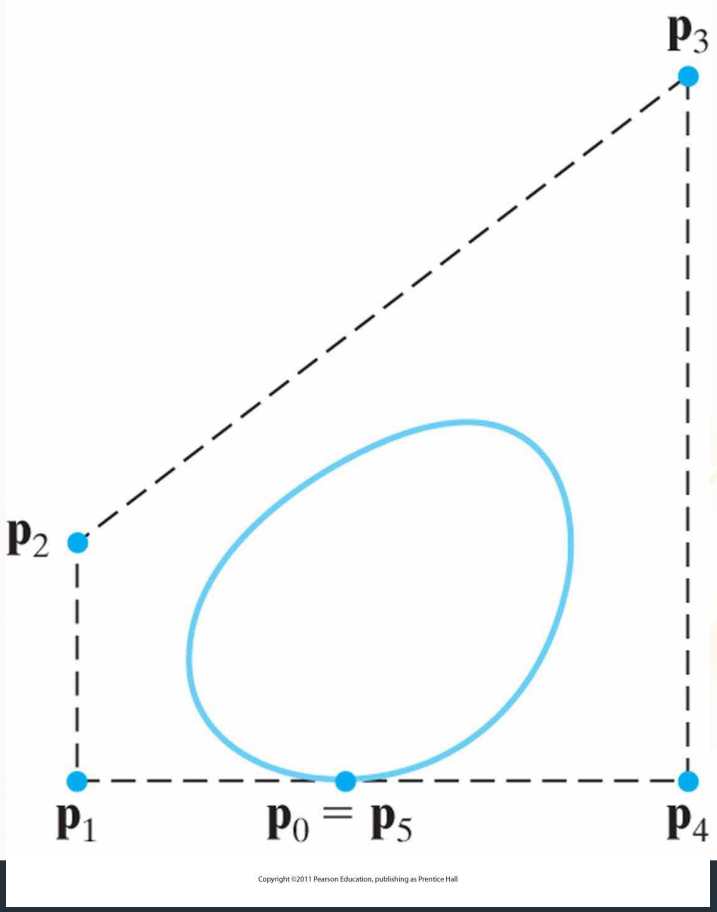
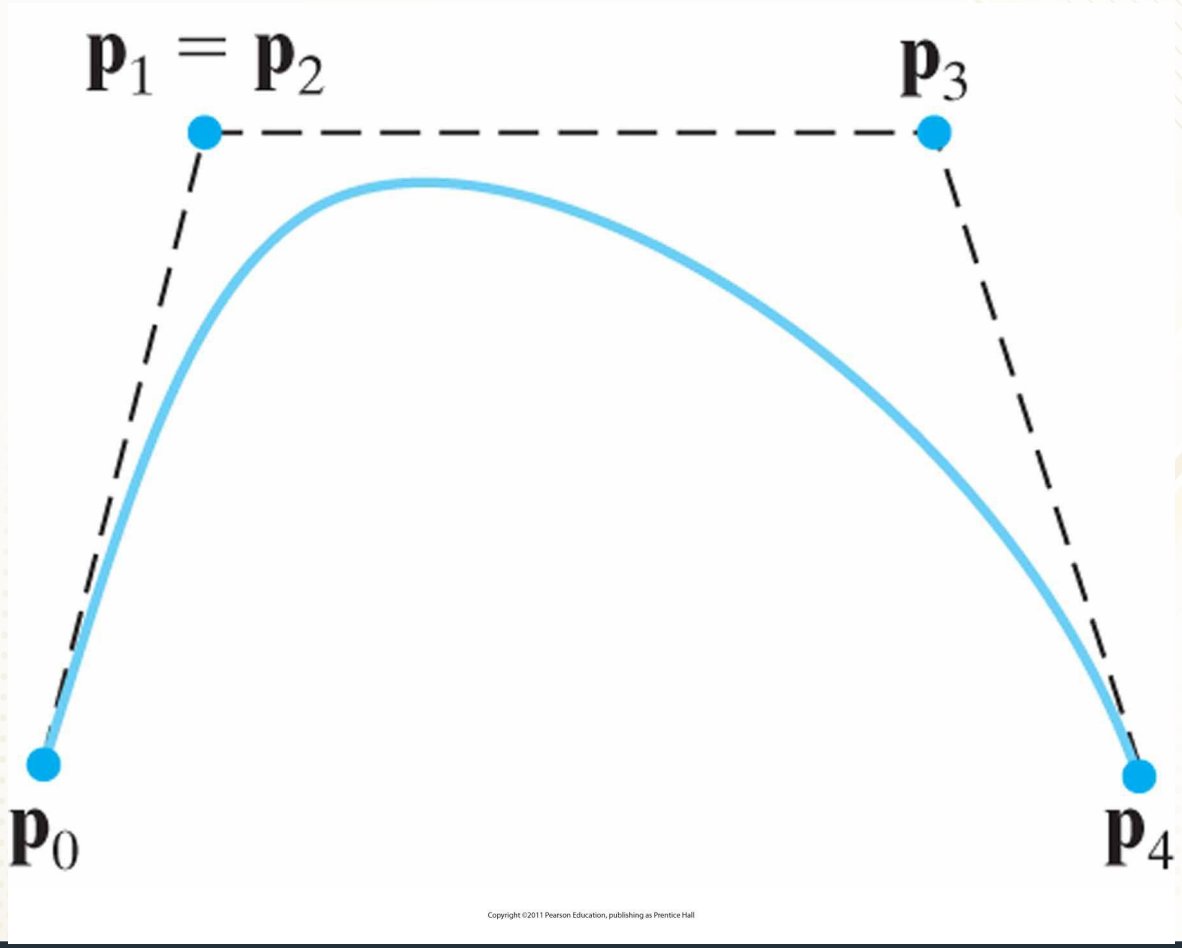


Figure 14-23 A Bézier curve can be made to pass closer to a given coordinate position by assigning multiple control points to that position.



Design Technique using Bezier curves

A closed Bézier curve is generated when we set the last control-point position to the coordinate position of the first control point.

Also, specifying multiple control points at a single coordinate position gives more weight to that position.

A single coordinate position is input as two control points, and the resulting curve is pulled nearer to this position.

Because Bézier curves connect the first and last control points, it is easy to match curve sections(zero-ordercontinuity).

Also, Bézier curves have the important property that the tangent to the curve at an endpoint is along the line joining that endpoint to the adjacent control point.

Cubic Bezier Curves

Bezier curves with many points are expensive to compute due to high degree of polynomials. It is common to “patch” curves of four points and construct it from piecewise cubic curves.

$$\mathrm{BEZ}_{0,3} = (1-u)^3, \quad \mathrm{BEZ}_{1,3} = 3u(1-u)^2, \quad \mathrm{BEZ}_{2,3} = 3u^2(1-u), \quad \mathrm{BEZ}_{3,3} = u^3.$$

At u=0, the only non zero blending function is $BEZ_{0,3}$, which has the value 1.At u =1,the only non zero function is $BEZ_{3,3}(1) = 1$.

Event Types

Window: resize, expose, iconify

Mouse: click one or more buttons

Motion: move mouse

Keyboard: press or release a key

Idle: nonevent

Define what should be done if no other event is in queue

Callbacks

Programming interface for event-driven input

Define a *callback function* for each type of event the graphics system recognizes

This user-supplied function is executed when the event occurs

GLUT example: **`glutMouseFunc(mymouse)`**

mouse callback function

Fonts in GLUT

Previous method requires us to create all letters. We prefer to use an existing font, rather than to define our own. GLUT provides a few raster and stroke fonts.

We can access a single character from a **monotype** (evenly spaced) font by the function call:
glutStrokeCharacter(GLUT_STROKE_MONO_ROMAN, int character)

GLUT_STROKE_ROMAN provides proportionally spaced characters with a size of approximately 120 units maximum. Note that this may not be an appropriate size for your program, thus resizing may be needed.

We control the location of the first character using a translation. In addition, once each character is printed there will be a translation to the bottom right of that character.
Note that translation and scaling may affect the OpenGL state. It is recommended that we use the *glPushMatrix* and *glPopMatrix* as necessary, to prevent undesirable results.

Fonts in GLUT – cont.

Raster or bitmap characters are produced in a similar manner. For example:
glutBitmapCharacter(GLUT_BITMAP_8_BY_13, int character)

will produce an 8x13 character.
The position is defined directly in the frame buffer and are not subject to geometric transformations. A **raster position** will keep the position of the next raster primitive. This position can be defined using *Raster-Pos*( )*.

If a character have a different width, we can use the function
glutBitmapWidth(font, char)
To return the width of a particular character.

```
glRasterPos2i(rx, ry);
glutBitmapCharacter(GLUT_BITMAP_8_BY_13, k);
rx += glutBitmapWidth(GLUT_BITMAP_8_BY_13, k);
```

Text and Display lists

1.Stroke character

- ***glutStrokeCharacter(GLUT_STROKE_MONO_ROMAN, int character)***

2. Raster character

- Font -8 x 13 pattern of bits. It takes 13 bytes to store each character
- ***glutBitmapWidth(font, char)***

```
glRasterPos2i(rx, ry);
glutBitmapCharacter(GLUT_BITMAP_8_BY_13, k);
rx += glutBitmapWidth(GLUT_BITMAP_8_BY_13, k);
```

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Programming Event Driven Input

Using the Pointing Device

Window Events

Keyboard Events

The Display and Idle Callbacks

Window Management

Two types of events

- A **move event** is generated when the mouse is moved with one of the buttons depressed. If the mouse is moved without a button being held down, this event is classified as a **passive move event**.
- A **mouse event** occurs when one of the mouse buttons is either depressed or released. A button being held down does not generate an event until the button is released.

- We want the depression of the left mouse button to terminate the program. The required callback is the single-line function
- void mouse(int button, int state, int x, int y)

```
{
 if(button == GLUT_LEFT_BUTTON && state ==
GLUT_DOWN)
exit();
 }
```

- We specify the mouse callback function, usually in the main function, glutMouseFunc(mouse)

Draw a small box at each location on the screen where the mouse cursor is located at the time that the left button is pressed. A push of the middle button terminates the program.

Main Program

```
int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("square");
    myinit();
    glutReshapeFunc(myReshape);
    glutMouseFunc(mouse);
    glutDisplayFunc(display);
    glutMainLoop();
```

- The mouse callbacks are again in the function mouse.

```
void mouse(int btn, int state, int x, int y)
{
 if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
drawSquare(x,y);
 if(btn==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN)
exit();
 }
```

Reshape event

- Generated whenever the window is resized, such as by a user interaction.

Window Events

Window event

- Most window systems allow a user to resize the window, usually by using the mouse to drag a corner of the window to a new location.
- Our **reshape function** probably should make sure that no shape distortions occur.
- If we want to redraw the objects that were in the window before it was resized, we need a mechanism for storing and recalling them.

```
void myReshape(GLsizei w, GLsizei h)
 {
    /* adjust clipping box */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    /* adjust viewport and clear */
    glViewport(0,0,w,h);
    glClearColor (0.0, 0.0, 0.0, 0.0); glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
```

Clear the screen each time that it is resized, and we use the entire new window as our drawing area.

The reshape event returns in its measure the height and width of the new window.

We use these values to create a new OpenGL clipping window using gluOrtho2D, as well as a new viewport with the same aspect ratio.

We then clear the window to black.

Keyboard Events

Keyboard events are generated when the mouse is in the window and one of the keys is depressed.

The ASCII code for the key pressed and the location of the mouse are returned.

All the keyboard callbacks are registered in a single callback function, such as

glutKeyboardFunc(keyboard);

- For example, if we wish to use the keyboard only to exit the program, we can use the callback function

```
void keyboard(unsigned char key, int x, int y)
{
 if(key==’q’ || key == ’Q’) exit( );
}
```

The Display and Idle Callbacks

glutDisplayFunc(display);

It is invoked when GLUT determines that the window should be redisplayed.

When the window is opened initially.

Display GLUT requires every program to have a display callback.

glutPostRedisplay()

- Avoids extra or unnecessary drawings of the screen by setting a flag inside GLUT's main loop indicating that the display needs to be redrawn.
- glutPostRedisplay ensures that the display will be drawn at most once each time the program goes through the main loop.

idle callback

- Invoked when there are no other events. Its default is the null function. A typical use of the idle callback is to continue to generate graphical primitives through a display function while nothing else is happening.

Window Management

GLUT also supports both multiple windows and subwindows of a given window. We can open a second top-level window by

id=glutCreateWindow("second window");

There turned integer value allows us to select this window as the current window into which objects will be rendered by

glutSetWindow(id);

Menus- Pop Up Menus

We must define actions corresponding to each entry in the menu.

We must link the menu to a particular mouse button.

Finally, we must register a callback function for each menu.

We can demonstrate simple menus with the example of a pop-up menu that has three entries.

- The first selection allows us to exit our program.
- The second and third change the size of the squares in our drawSquare function.
- We name the menu callback demo_menu.
- The function calls to set up the menu and to link it to the right mouse button should be placed in our main function.

Menus

Exit
Increase Square Size
Decrease Square Size

```
glutCreateMenu(demo_menu);
glutAddMenuEntry("quit",1);
glutAddMenuEntry("increase square size", 2);
glutAddMenuEntry("decrease square size", 3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

The second argument in each entry's definition is the identifier passed to the callback when the entry is selected

```
void demo_menu(int id)
{
      if(id == 1) exit( );
      else if (id == 2) size = 2 * size;
      else if (size > 1) size = size/2;
      glutPostRedisplay( );
}
```

Sub menu

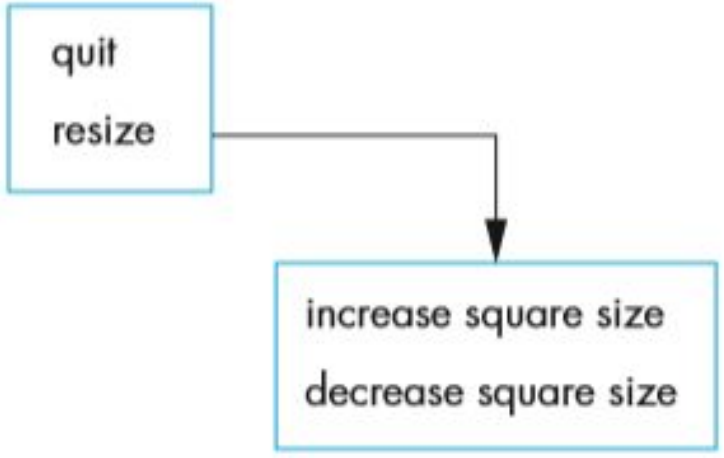


Figure 3.17 Structure of hierarchical menus.

Sub Menu

- GLUT also supports hierarchical menus.

Example,

- Suppose that we want the main menu that we create to have only two entries.
- The first entry still causes the program to terminate.
- Second causes a submenu to pop up.
- The submenu contains the two entries for changing the size of the square in our square-drawing program.

The following code for the menu (which is in main) should be clear:

```
sub_menu = glutCreateMenu(size_menu);
glutAddMenuEntry("Increase square size", 2);
glutAddMenuEntry("Decrease square size", 3);
glutCreateMenu(top_menu);
glutAddMenuEntry("Quit",1);
glutAddSubMenu("Resize", sub_menu);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

Writing the callback functions, size_menu and top_menu, should be a simple exercise for you

Building Interactive Models

Using OpenGL, we can develop a program where we can do insertion, manipulation, deletion etc and we can also build a program which is quite interactive by using the concept of instancing and display lists.

We want to build a complex image using a set of predefined building blocks. These can be simple geometric shapes.

Consider in interior design application which has items like chairs, tables and other house hold items. These items are called the basic building blocks of the application.

Double Buffering

When we redisplay our CRT, we want to do so at a rate sufficiently high (50-80 times/sec) that we cannot notice the clearing and redrawing of the screen.

That means the contents of the frame buffer must be drawn at this rate.

One problem may occur when a complex shape is drawn. In that case the display may not be done in one refresh cycle. A moving object will be distorted.

Double buffering can provide a solution to these problems. The **front buffer** is displayed when we draw in the **back buffer**. We can swap the back and front buffer from the application program.

With each swap, the display callback is invoked.

The double buffering is set using the *GLUT_DOUBLE,* instead of *GLUT_SINGLE* in *glutInitDisplayMode.* The buffer swap function using GLUT will be: *glutSwapBuffer( );*

Double buffering does not speed of the process of displaying a complex display. It only ensures that we never see a partial display.

Timer Function

Put delays in application program

`glutTimerFunc` registers a timer callback to be triggered in a specified number of milliseconds.

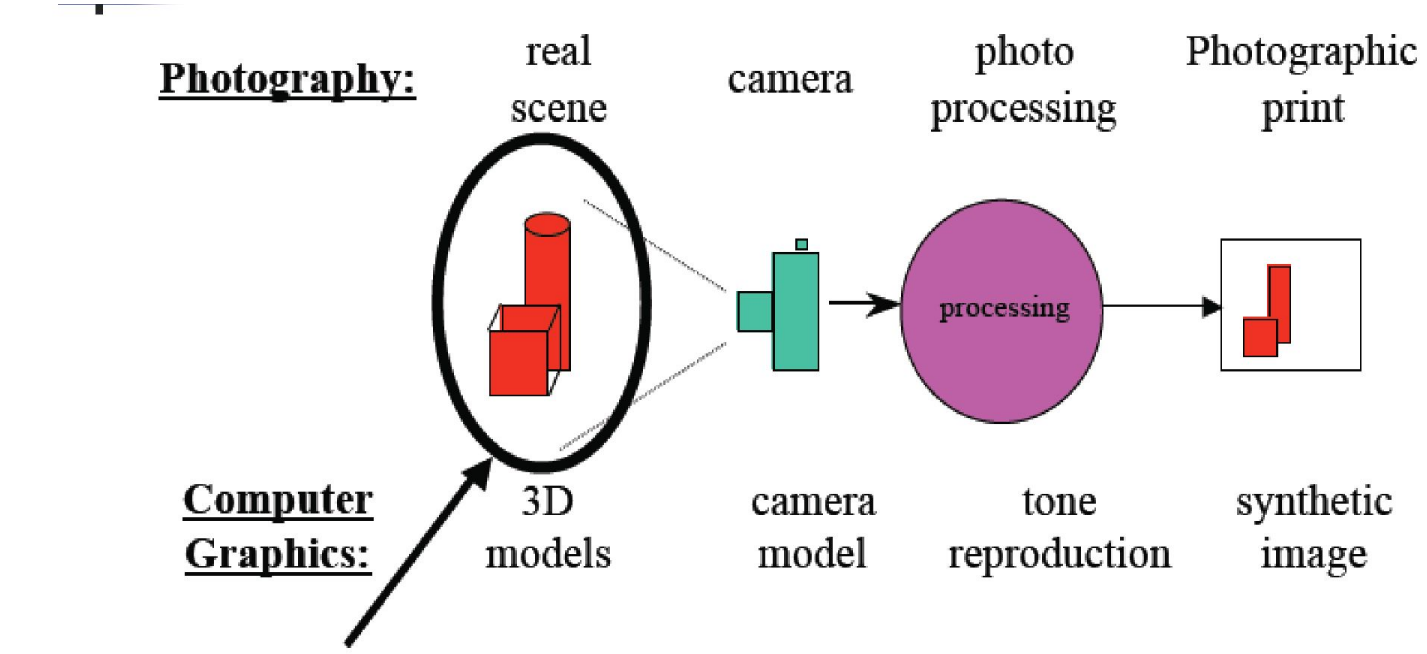
void glutTimerFunc(int delay, void (*func)(int), int value);

Starts timer in the event loop that delays the event loop for delay in milliseconds

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

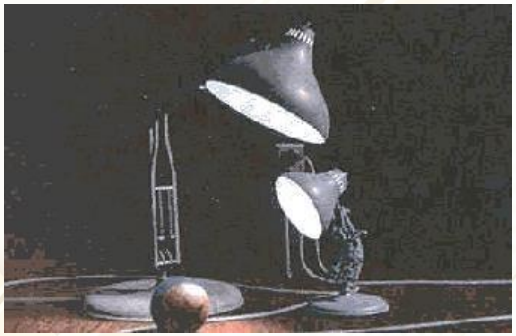
What is Animation

- A motion picture comprises a series of drawings/images simulating motion by means of slight progressive changes in the drawings
- Computer animation?
  - Make objects change over time according to scripted actions, and the animation consists of relative movement between rigid bodies and possibly movement of the virtual camera (or view point)
  - Using a standard renderer to produce consecutive frames



Computer Animation

- **Animation Pipeline**
  - 3D modeling
  - Motion specification
  - Motion simulation
  - Shading, lighting, & rendering
  - Postprocessing

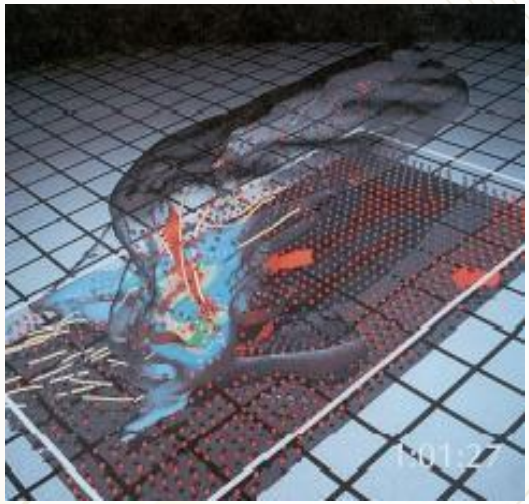


Computer Animation

- This is completely analogous to model animation where scale models are photographed by special cameras

Simulation

- **What is simulation?**
  - Predict how objects change over time according to physical laws



Why Animation Works?

- The eye cannot register images faster than approximately 50 frames per second (30 is just about adequate)
- If a gap in the projection occurs, the eye seems to perform spatial interpolation over the gap

Computer Animation

- Creating animation sequences
  - object definition
  - path specification
  - key frames
  - in-betweening
- Parametric equations
- Displaying animation sequences
  - raster animation

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Traditional Animation Techniques

- Prior to the advent of computational era……

Overview: Traditional Animation

- **Early 2D Animation:** Used traditional techniques
- **Early 3D Animation:** Neglected traditional techniques
- Understanding of these **Fundamental principles of** traditional animation techniques is essential to producing good computer animation

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Basic Principles of Traditional Animation (from Disney)

- Squash and stretch
- Slow in and out
- Anticipation
- Exaggeration
- Follow through and overlapping action
- Timing
- Staging
- Straight ahead action and pose-to-pose action
- Arcs
- Secondary action
- Appeal

Squash and Stretch

- Style vs. accuracy
- Teaches basic mechanics of animation
- Defines rigidity of material
- Time interpolation captures accuracy of velocity
- Squash and stretch replaces motion blur stimuli and adds life-like intent

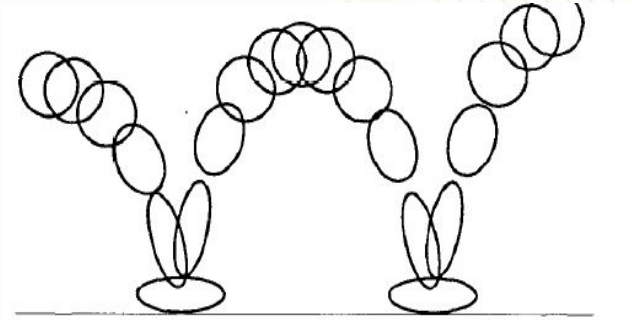
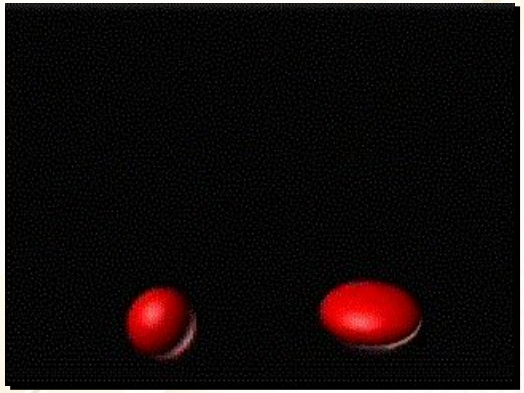
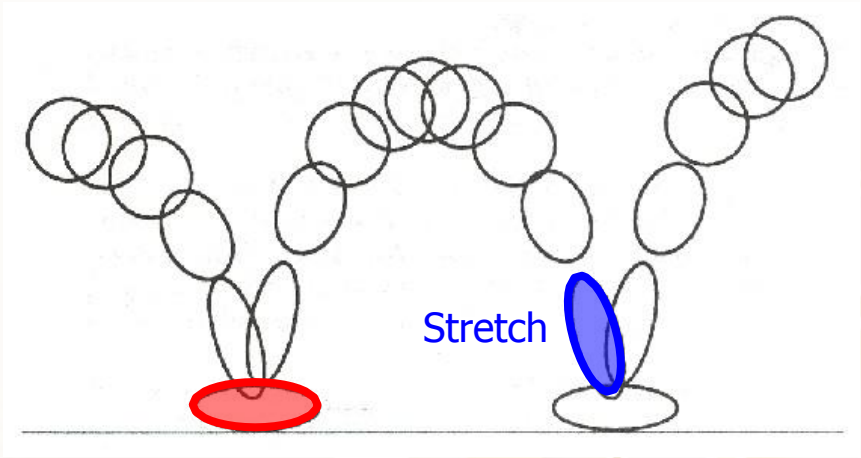


FIGURE 2. Squash & stretch in bouncing ball.



RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Squash and Stretch



Squash and Stretch

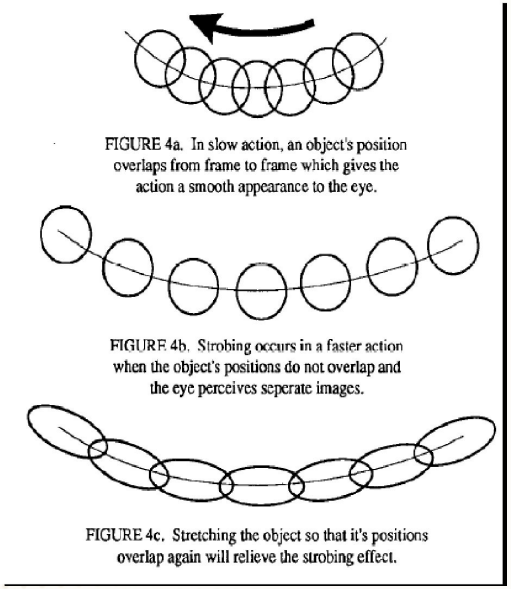


FIGURE 4a. In slow action, an object's position overlaps from frame to frame which gives the action a smooth appearance to the eye.

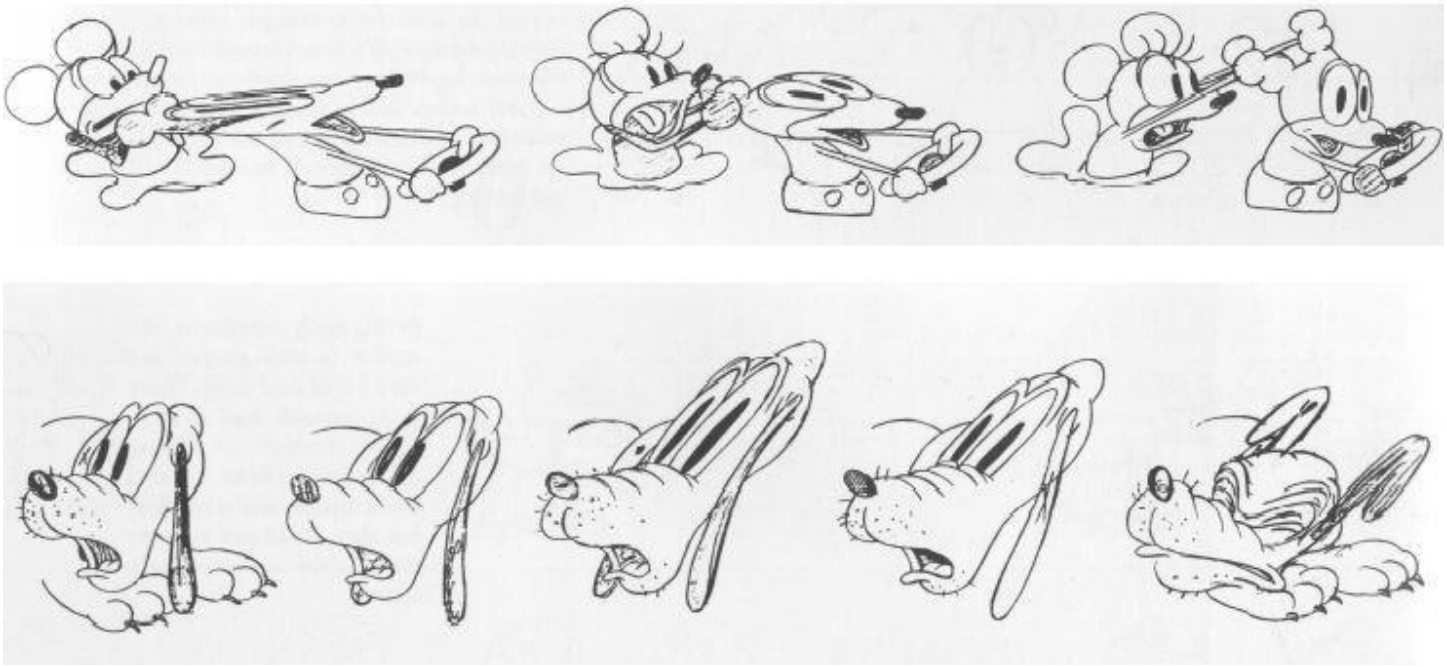
FIGURE 4b. Strobing occurs in a faster action when the object's positions do not overlap and the eye perceives seperate images.

FIGURE 4c. Stretching the object so that it's positions overlap again will relieve the strobing effect.

- Can relieve the disturbing effect of strobing

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Squash and Stretch



Timing and Motion

- Gives meaning to movement
- Proper timing is critical to making ideas readable

•Examples:

2. Timing: tiny characters move quicker than larger ones.

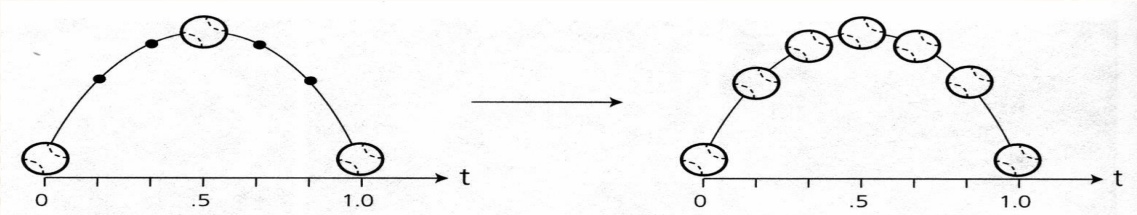


Figure 10.10 **Inbetweening with nonlinear interpolation and easing.** The ball changes speed as it approaches and leaves keyframes, so the dots indicating calculations made at equal time intervals are no longer equidistant along the path.

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Anticipation

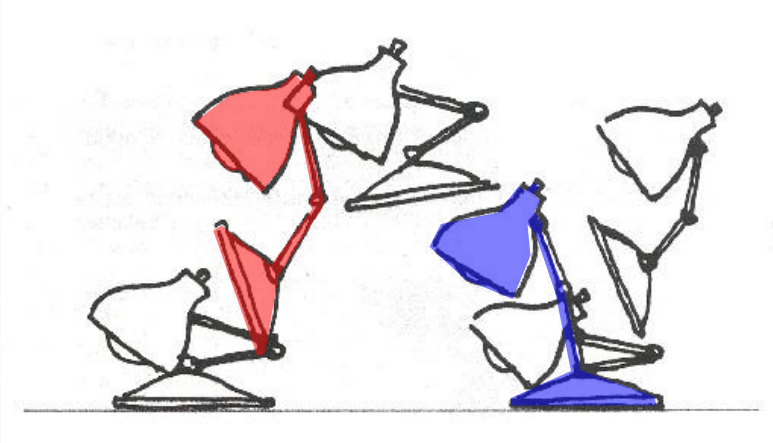
Preparation for an action

Example:

- Goofy prepares to hit a baseball.

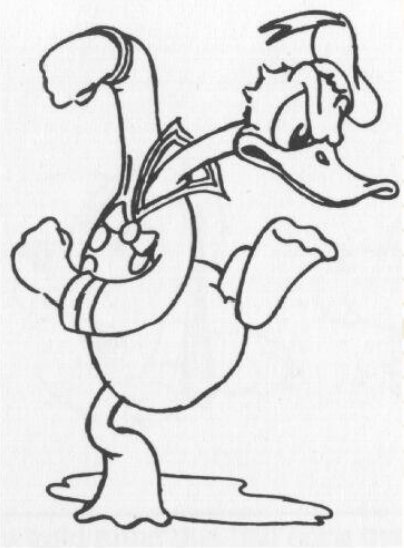
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Anticipation



Staging

- A clear presentation of an idea
- Don't surprise the audience
- Direct their attention to what's important

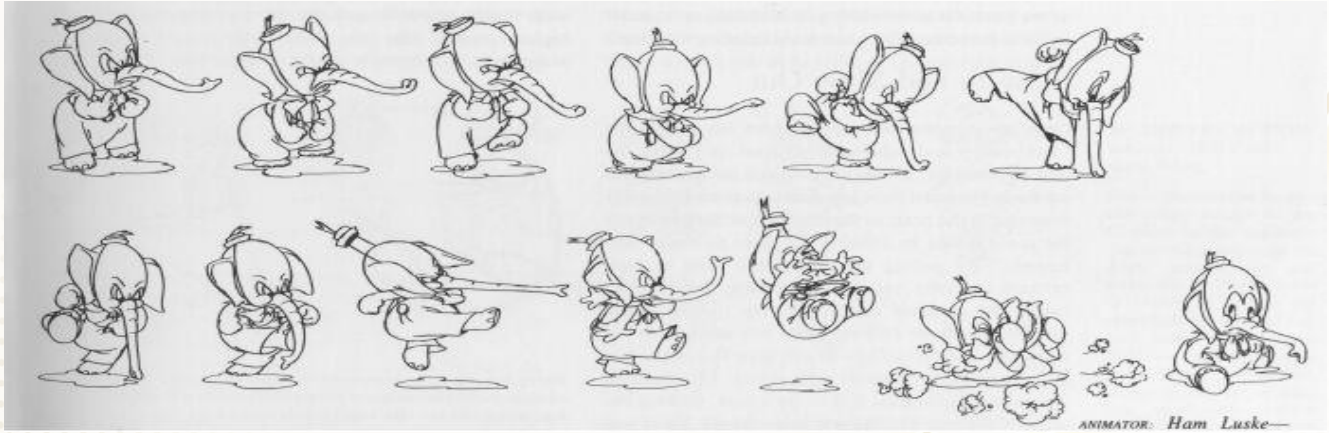


Some Techniques:

1. **Use motion in a still scene or use of static movement in a busy scene.**
2. **Use of** silhouettes (to the side)

Follow Through

- Audience likes to see resolution of action
- Discontinuities are unsettling



Follow Through and Overlapping Action

Follow Through

Termination part of an action

Example: after throwing a ball

2. Overlapping Action

Starting a second action before the first has completed.

Example: Luxo Jr.'s hop with overlapping action on chord

Secondary Motion

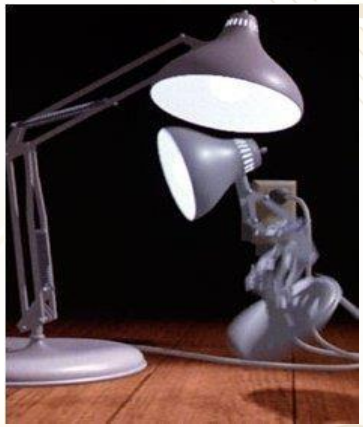
- Characters should exist in a real environment
- Extra movements should not detract

Secondary Action

- Action that results directly from another action
- Used to increase the complexity and interest of a scene

Example:

Body movement is the primary action, facial expression is the secondary action

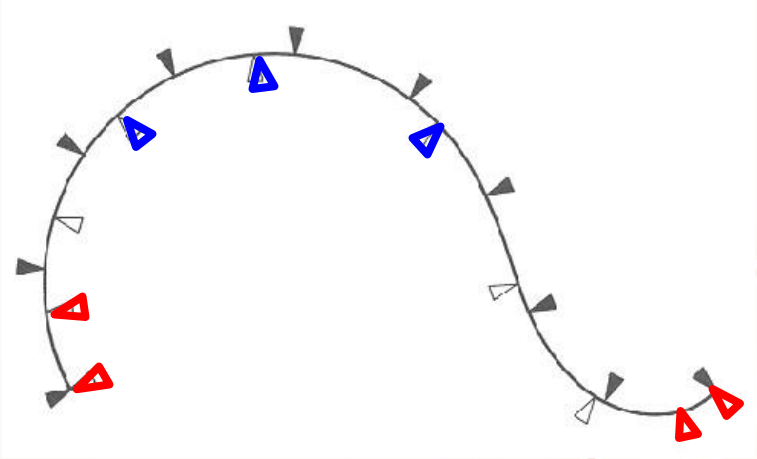


Straight Ahead Action and Pose-to-Pose Action

- **Straight Ahead**
  - Animator starts from first drawing in the scene and draw all subsequence frames until the end of scene.
- **Pose-to-Pose**
  - Animator plans actions, draws a sequence of poses, in between frames etc.

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Slow In and Out



Slow in and Out

Spacing of inbetween frames to achieve subtlety of timing and movement.

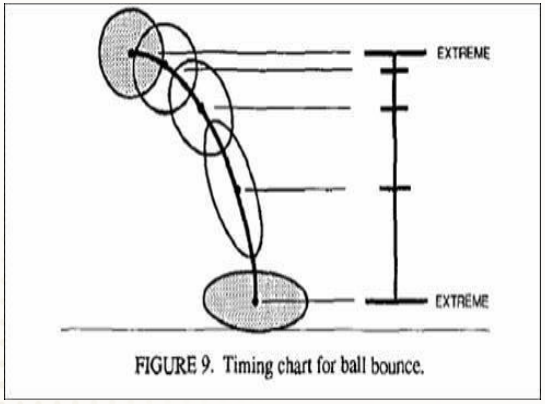


FIGURE 9. Timing chart for ball bounce.

1. 3d keyframe comp. Systems uses spline interpolation to control the path of an object.

2. Has tendency to overshoot at extremes (small # of frames).

Arcs

- Visual path of action for natural movement
- Makes animation much smoother and less stiff than a straight line

Exaggeration

- Emphasizing the essence of an idea via the design and the action
- Needs to be used carefully

Example: Luxo Jr. made smaller to give idea of a child.

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Appeal

- Refers to what an audience would like to see
- Character cannot be too simple (boring) or too complex

Examples:

Avoid mirror symmetry, assymmetry is interesting.

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

What techniques used for Wally B.?

What do you think Wally B's going to do?

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

The Action: Zooooooooooommmm!

Termination: Poof! He’s gone!

Role of Personality

- Animator's first goal is to entertain
- Success of animation lies in the personality of the characters

Conclusion

Hardware/Software are simply not enough, these principles are just as important tools too.

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Basic Steps for a Simple Computer Animation

1. Creating animation sequences
   - Object/model definition
   - path specification (for an object or a camera)
   - key frames
   - in-betweening
- 2. Displaying the sequences

Displaying Animation Sequences

- Movies work by fooling our eyes
- A sequence of static images presented in a quick succession appears as continuous flow

Displaying Animation Sequences

- To achieve smooth animation, a sequence of images (frames) have to be presented on a screen with the speed of at least 30 per second
- Animations frames can be
  - pre-computed in advance and pre-loaded in memory
  - computed in real time (e.g. movement of the cursor)

Raster Animation

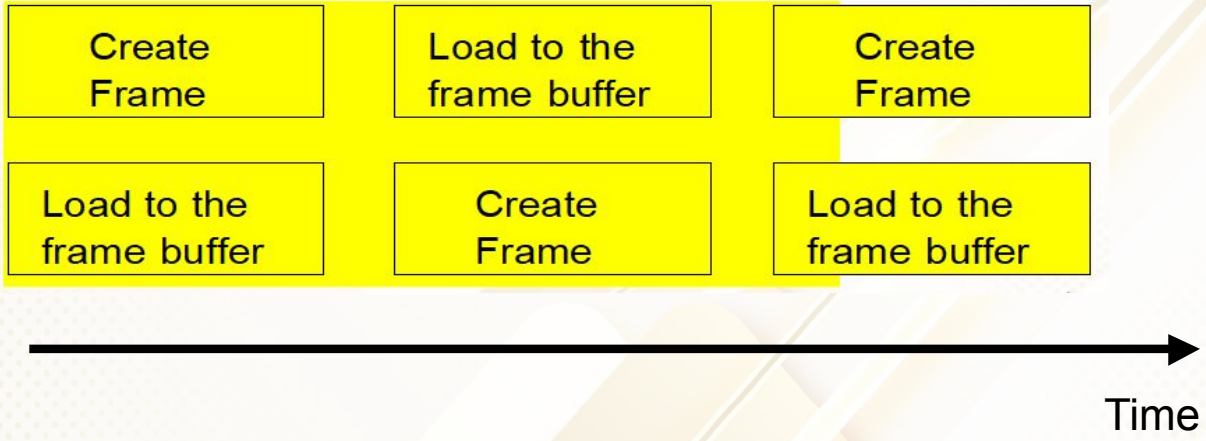
- This is the most common animation technique
- Frames are copied very fast from off-screen memory to the frame buffer
- Copying usually done with bit-block-transfer-type operations
- Copying can be applied to
  - complete frames
  - only parts of the frame which contain some movement

Raster Animation - Procedures

- A part of the frame in the frame buffer needs to be erased
- The static part of the frame is re-projected as a whole, and the animated part is over-projected.

Double Buffering

- Used to achieve smooth animation
- The next frame of animation is computed to an off-screen buffer at the same time when the current frame is transferred to the frame buffer.



Creation of Animation Sequence Object Definition

- In simple manual systems, the objects can be simply the artist drawings
- In computer-generated animations, models are used
- Examples of models:
  - a "flying logo" in a TV advertisement
  - a walking stick-man
  - a dinosaur attacking its prey in Jurassic Park

Models Can Be

- Rigid (i.e., they have no moving parts)
- Articulated (subparts are rigid, but movement is allowed between the sub-parts)
- Dynamic (using physical laws to simulate the motion)
- Particle based (animating individual particles using the statistics of behavior
- Behavior based (e.g., based on behavior of real animals)

Rigid Objects

- Simple rigid objects can be defined in terms of
  - Basic shapes such as line segments, circles, splines etc. (2D)
  - Polygon tables (3D)

- Rigid body animation is an extension of the three-dimensional viewing

Rigid Body Animation

- Rigid body animation uses standard 3D transformations
- At least 30 frames per second to achieve smooth animation
- Computing each frame would take too long

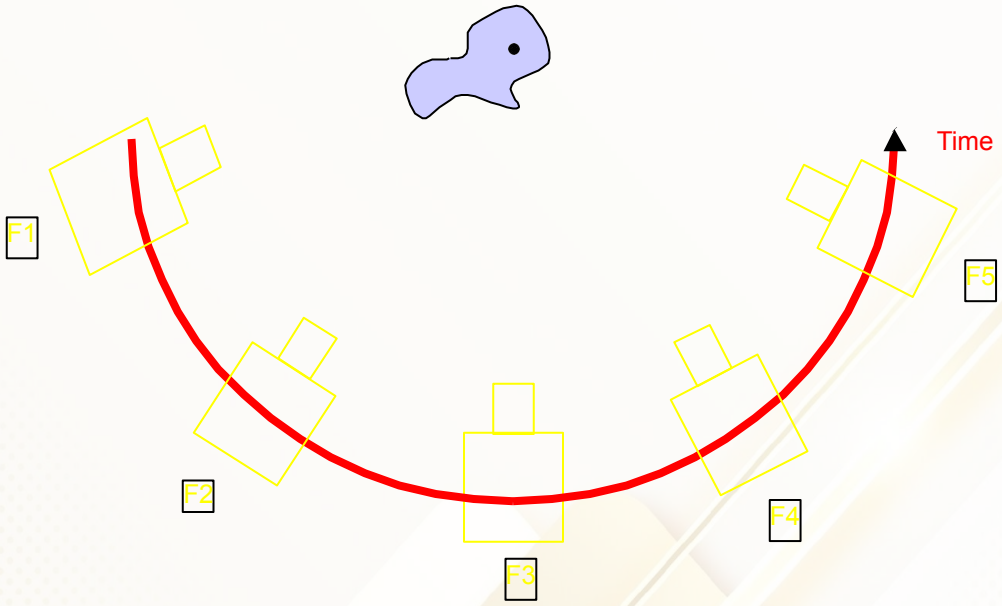
Path Specification

- Impression of movement can be created for two basic situations, or for their combination:
  - static object, moving camera
  - static camera, moving object
- The path defines the sequence of locations (for either the camera or the object) for the consecutive time frames

Static Object, Moving Camera

- The path specifies the spatial coordinates along which the camera moves
- The path is usually specified for a single point, e.g. the view reference point

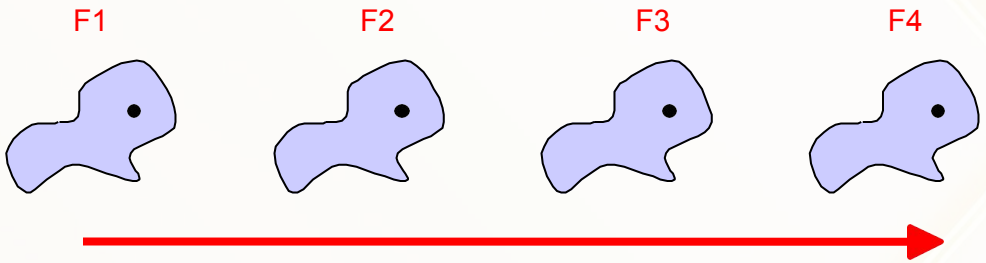
Static Object, Moving Camera

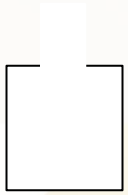


Static Object, Moving Camera

- During movement, the target point in the world coordinate system can
  - remain the same (e.g., when walking or flying around the object to see it from all directions);
  - change (e.g., standing in one location and looking round, or moving along a given path and showing the view seen by the observer while moving).

Static Camera, Moving object





Static Camera, Moving object

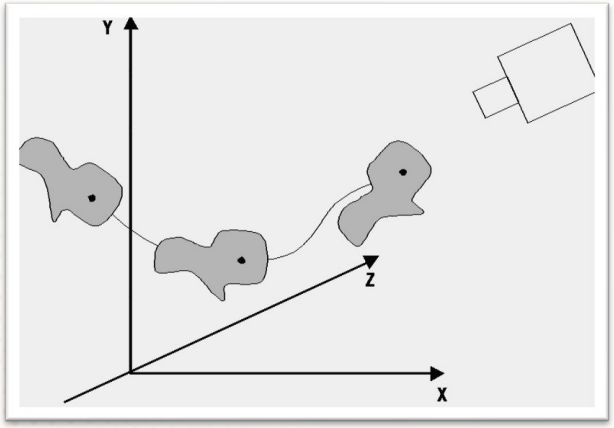
- Path specifying the object movement has to be defined

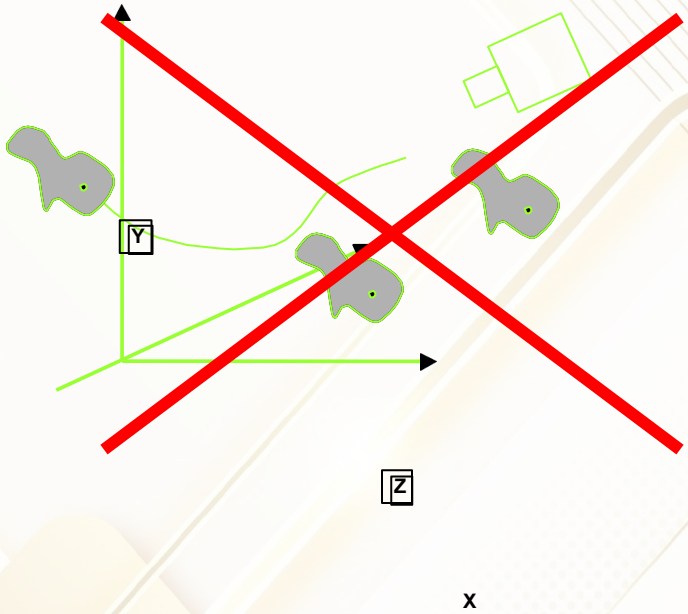
- The path is defined as the spatial coordinates along which the object moves

Static Camera, Moving Object

- Objects and their parts are defined in a local coordinate system
- Animation path is defined in the world coordinate system
- The path is specified for a single point, e.g., the center of the object's local coordinate system
- Coordinates of the actual points describing the object are calculated afterwards

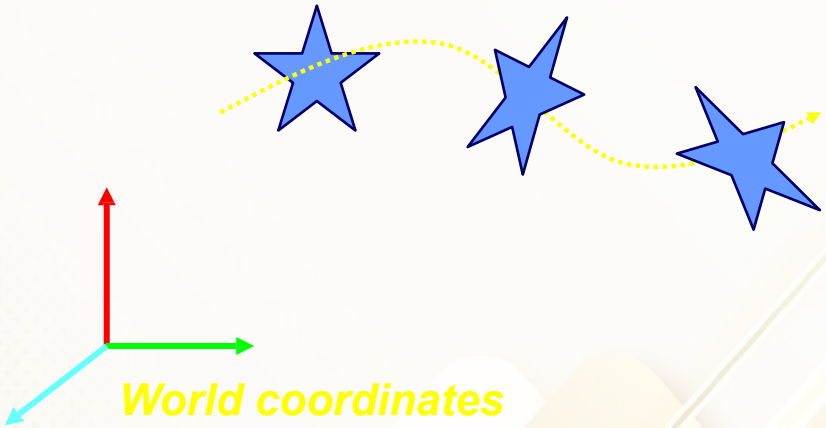
It is important to remember that when the object moves along the path, not only its position changes, but also its orientation





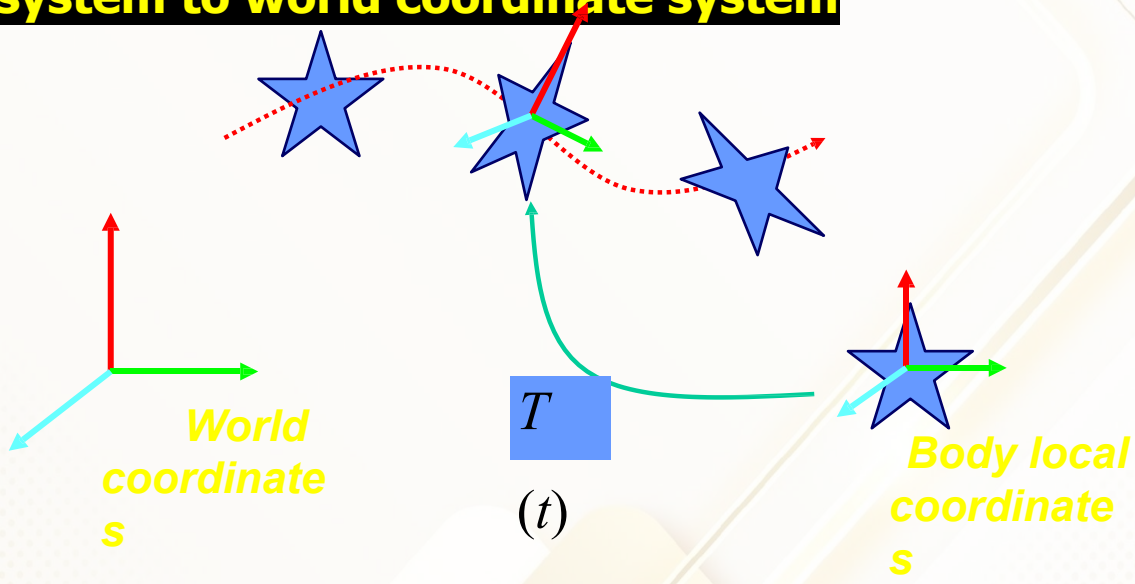
Motion

- Motion is *a time-varying transformation* from body local system to world coordinate system (in a very narrow sense)

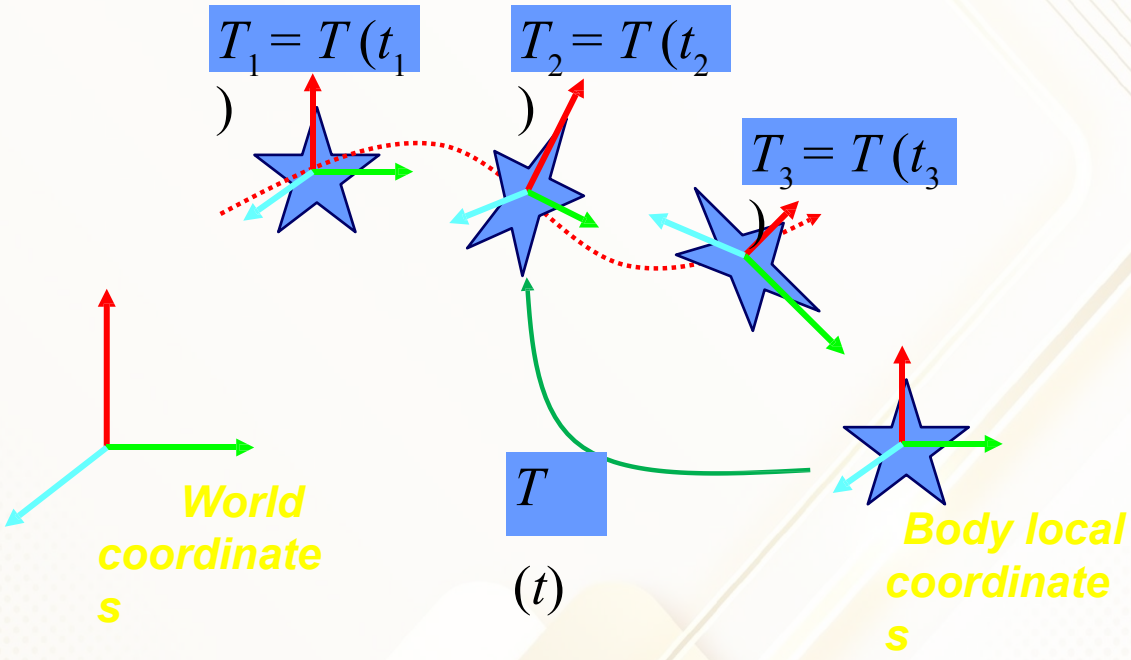


Keyframes and Inbetweening _Motion

Motion is a time-varying transformation from body local system to world coordinate system



Keyframing


$T_1 = T(t_1)$
$T_2 = T(t_2)$
$T_3 = T(t_3)$
T
(t)
World coordinates
Body local coordinates

Keyframing

- *Keyframe* systems take their name from the traditional hierarchical production system first developed by Walt Disney
- Skilled animators would design or choreograph a particular sequence by drawing frames that establish the animation – these are the so-called keyframes
- The production of the complete sequence is then passed on to less skilled artists who used the keyframes to produce 'in-between' frames

Key framing

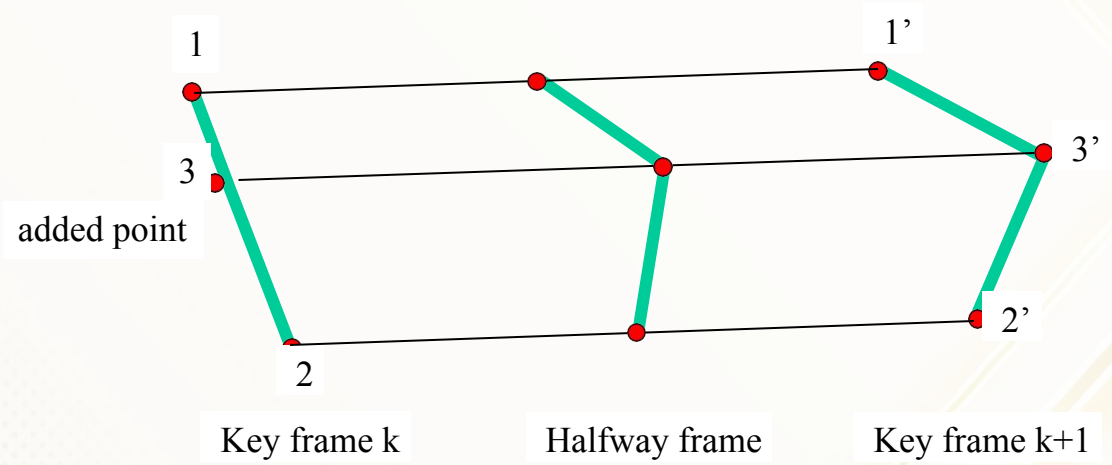
- The emulation of this system by the computer, whereby interpolation replaces the inbetween artist, was one of the first computer animation tools to be developed
- This technique was quickly generalized to allow for the interpolation of any parameter affecting the motion
- Special care must be taken when parameterizing the system, since interpolating naive, semantically inappropriate parameters can yield inferior motion

In-betweening

- The simplest method of in-betweening is linear interpolation
- Interpolation is normally applied to the projected object points

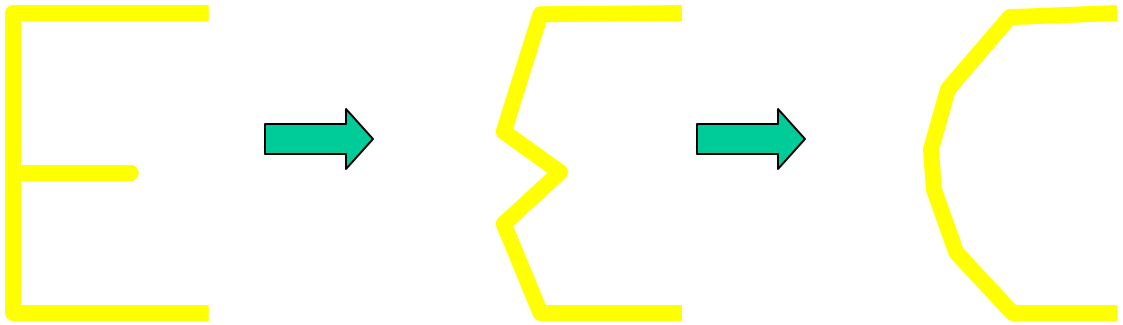
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Example



Example

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS



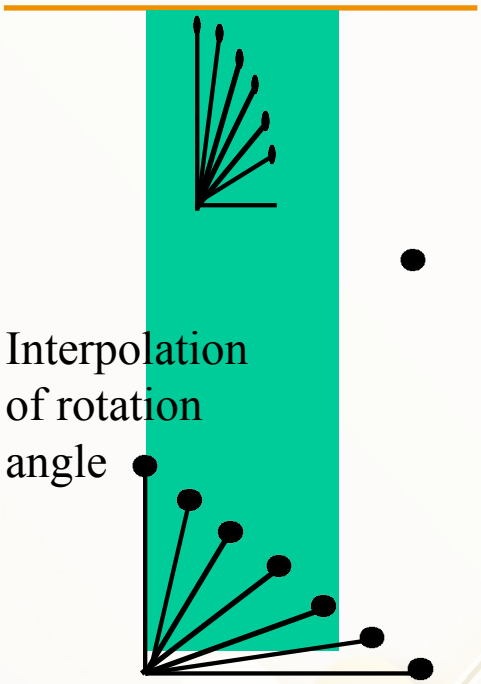
Key Frames

- First compute a small number of key frames
- Interpolate the remaining frames in-between these key frames (in-betweening)
- Key frames can be computed
  - at equal time intervals
  - according to some other rules
  - for example when the direction of the path changes rapidly

Key Frames

- The keyframing approach carries certain disadvantages
  - First, it is only really suitable for simple motion of rigid bodies
  - Second, special care must be taken to ensure that no unwanted motion excursions are introduced by the interpolant
  - Nonetheless, interpolation of key frames remains fundamental to most animation systems

Key Frame



In-betweening

- Linear interpolation will not always produce realistic results

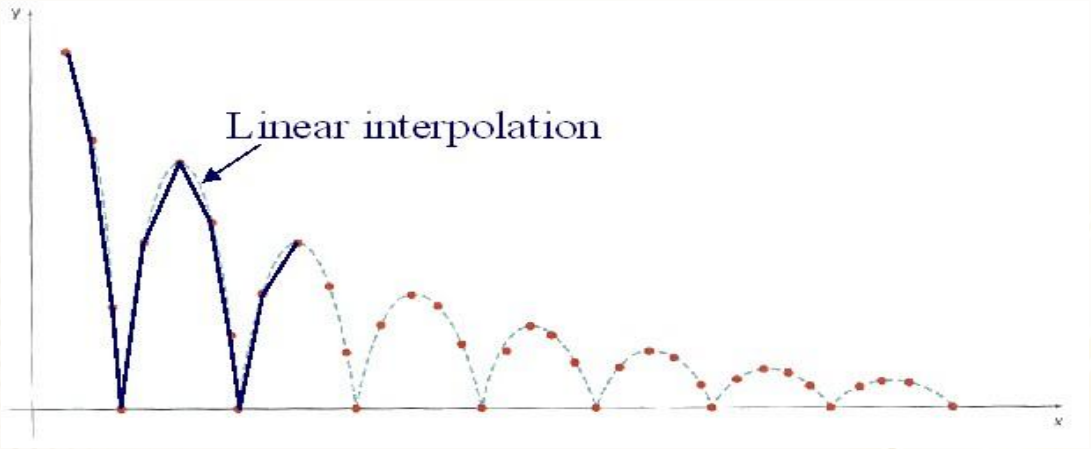
- Example: an animation of a bouncing ball where the best in-betweening can be achieved by dynamic animation

In-betweening

- In-betweening should use interpolation based on the nature of the path, for example:
  - straight path (linear interpolation)
  - circular path (angular interpolation)
  - irregular path linear interpolation, spline

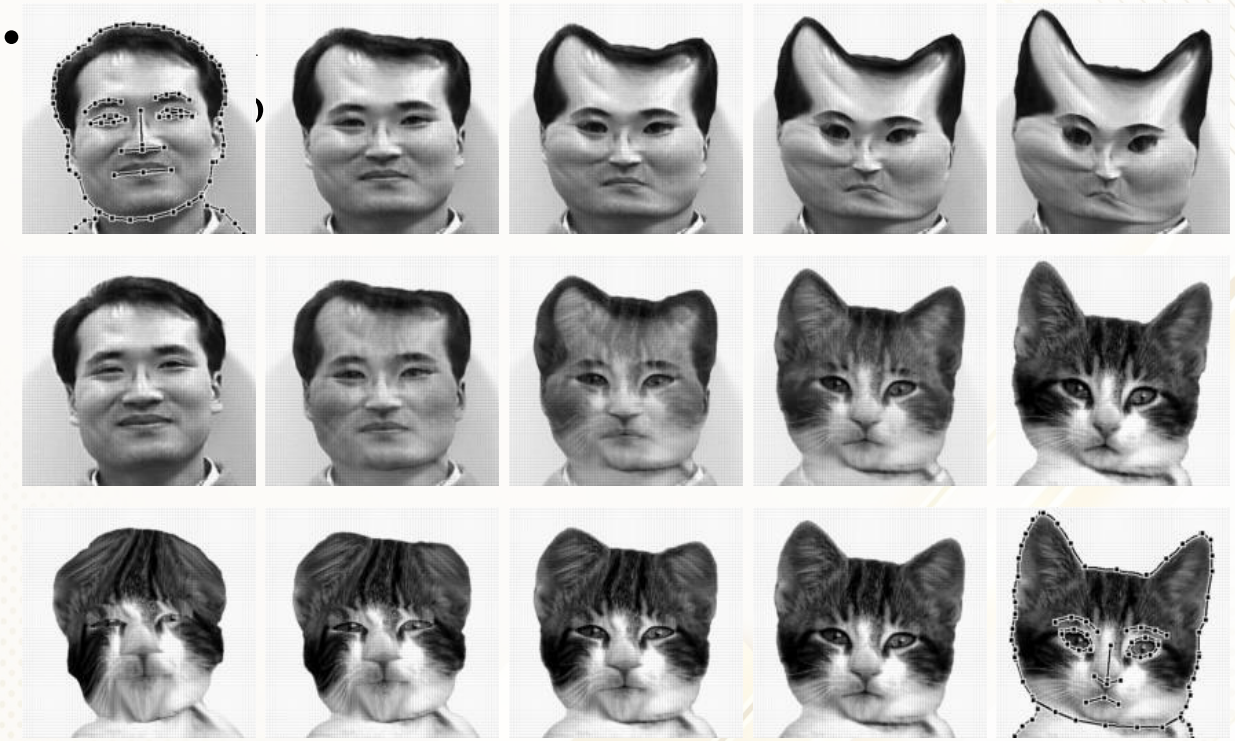
In between Frames

- **Linear Interpolation**
  - Usually not enough continuity



RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Image Morphing

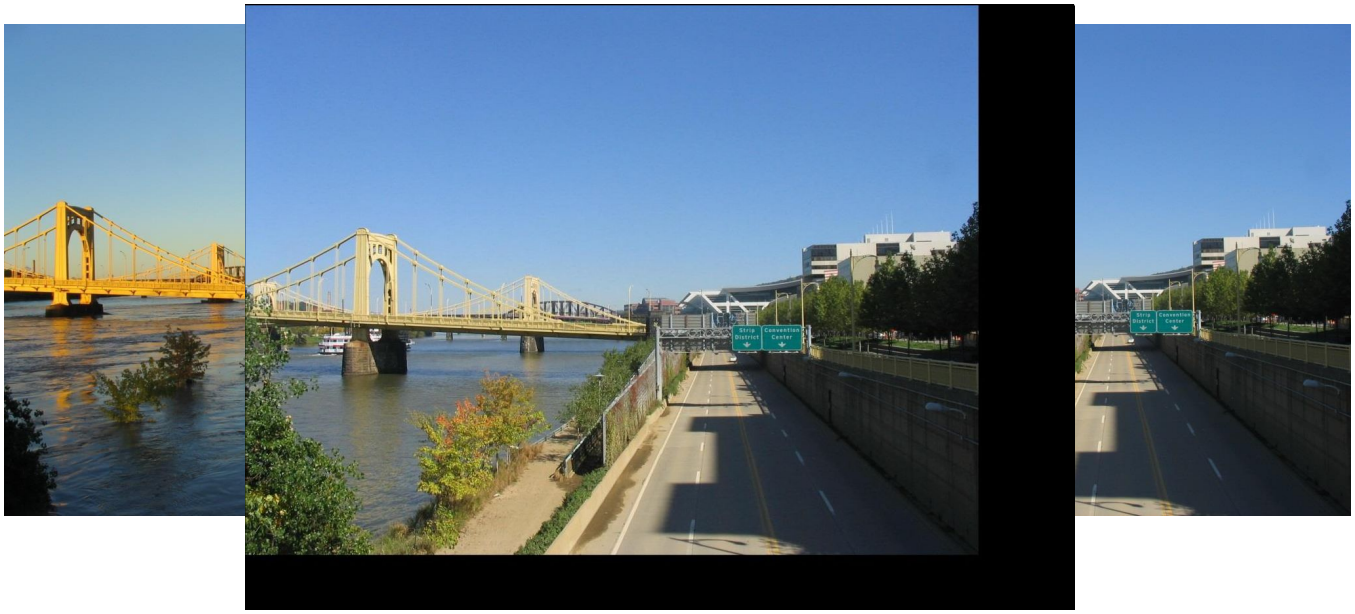
- 

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Image Morphing

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Image Morphing



RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

View Morphing

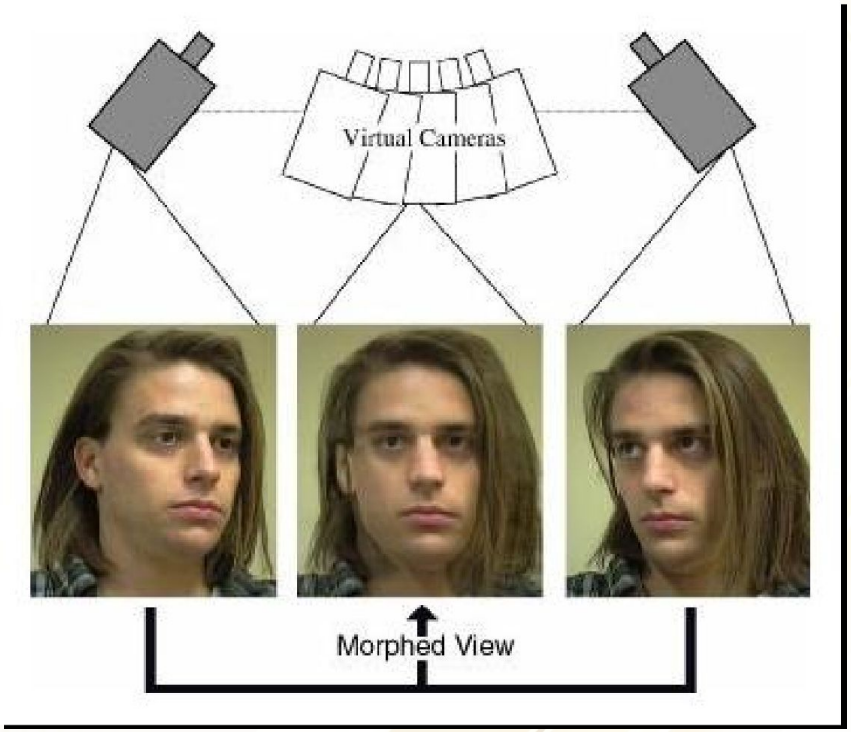
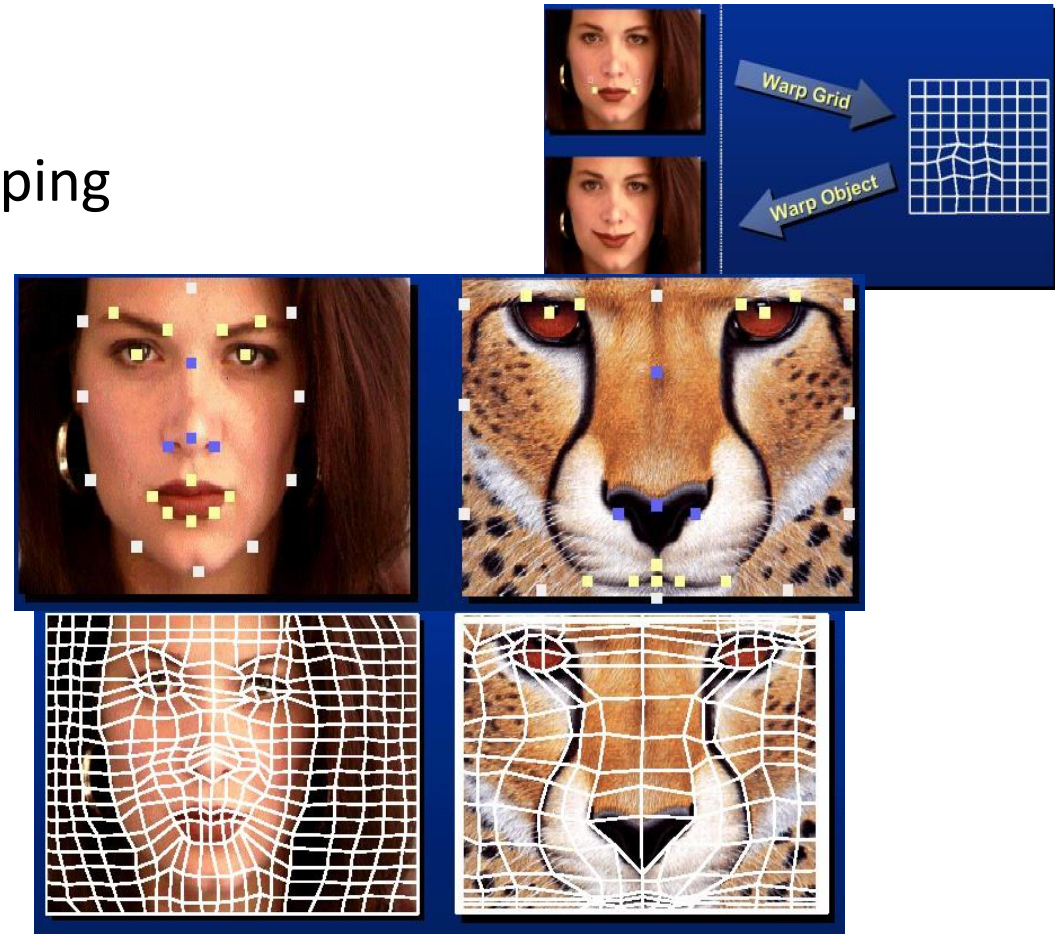


Image Warping



Animation Challenges

- Animating one rigid object with 6 degrees of freedom for 5 seconds at 30 frames per second requires 9000 numbers to be interactively specified
- A fully defined human figure will have more than 200 degrees of freedom
- A control hierarchy reduces the number of degrees of freedom (DOFs) that the animator has to specify
  - High-level constructs are mapped to lower-level data