**CURVES**

Parametric form:

Different types of parameterizations:

**Arc-Length Parameterization:**

Parameter represents arc length along curve

EXAMPLE:

To check if a curve is ALP:

**Unit Parameterization:**

Parameter represents a specific unit (mostly of time)

**CONTINUITY**

A **continuous** function is a function you can draw without lifting a pen

There are **different measures** of continuity:

continuous means:

All derivatives are continuous

: Positions are cont.

: Positions and tangents are cont.

: Just imagine you’re driving a steering wheel along curve. If wheel stays in same direction throughout curve, it’s ; otherwise no.

Be careful of what the derivative actually is.

A-----B-----C = C(1) (actually C(inf))

A-------B---C = C(0) cause speeds are different

continuity **only** depends on direction, not speed, since both curves are converted to arc-length parameterizations.

**HERMITE FORM**

Simpler, more readable way of writing a cubic function; makes sure the coefficients actually mean something to us.

Simply lists the locations of the beginning and end points and their derivatives at that point

Assuming that the points are and the derivatives are : (next column)

**HERMITE EQUATION**:

Or, in matrix form:

A number and numbers on a white background

Description automatically generated

**BEZIER CURVES**

Used to approximate interpolation points when you don’t know exactly what it is.

Bezier curves are defined by a set of control points; curve interpolates first and last but does not usually pass through intermediate ones. However, position of intermediate points control shape of the curve.

The formula for a Bezier cubic is:

WHY 3? Short answer: math works out that way ☺

PROPERTIES OF BEZIER CURVES:

Any number of points

Polynomials of degree

They interpolate first/last points

1st derivative at begin/end comes from 2 points

1st derivative scaled by deg. of polynomial – it’s why coefficient is 3 for Bezier cubics

Bezier curves stay inside of their convex hull (stay inside shape drawn by all points)

Are affine invariant (you don’t have to transform all points of Bezier curve, just the control points)

Are symmetric

Are variation diminishing (the curve can only wiggle based on the number of control points)

**B-SPLINE CURVES**

A B-spline curve is defined by a set of control points and a set of basis functions. Unlike Bezier curves, which are controlled by their control points directly, B-spline curves are defined by a set of control points that influence the curve indirectly through these basis functions.

B-spline curves offer several advantages, including their ability to smoothly interpolate a set of control points, their local control property (modifying one control point affects only a local portion of the curve. B-Splines are useful for making C(2) continuity

**CAMERA**

In JavaScript, the position of an object in 3D space is measured by a xyz coordinate system. However, we also have a camera that we use to draw objects. The camera has three axes – u,v,w.

U is parallel to camera

V goes above camera

W goes behind camera

**World Coordinate System:** system we arbitrarily choose to conveniently describe stuff we want to draw. However, we need to use the camera to convert the world coordinates into something relative to the place we’re drawing from. Enter the **lookAt** transform.

lookAt(out, eye, center, up)

out: matrix output

eye: origin of camera coordinate system (in world coordinates)

target (center): Point viewer is looking at (tilts the camera)

up: up vector (vector that when viewed through camera is perfectly vertical)

**Remember** that the up vector is not the same as the v-axis of the camera system. (They may be in the same plane, but that doesn’t mean they’re identical.)

If you have a matrix for the lookAt transformation:

A group of letters on a dark background

Description automatically generatedSimply multiply by world coordinates to get camera coordinates. For the other way around, inverse the matrix and multiply by camera coordinates.

**DRAWING**

**context.save()**: used to save current state of drawing context

**context.restore()**: restores canvas context to state that was saved by the last context.save() operation

**context.translate(x, y):** used to translate (move) the canvas rendering context along x/y axes

**context.rotate()**: rotates canvas context by a certain degree

**context.scale(0.5,1)**: scales canvas context by 0.5 horizontally and 1 vertically

**HIERARCHICAL MODELING**

In this approach, objects are grouped into parent-child relationships, where transformations applied to a parent object are inherited by its child objects. When transformations are applied, they are relative to the object's local coordinates.