italic = specific elements of the software
gray = details specific to a particular *object type*

3D Modeling Reimagined

Next generation 3D modeling application.

The 3D environment consists of three intersecting, perpendicular axes, each with positive and negative values respectively extending from the origin, out to infinity along each of the axis's two directions.

Objects in the 3D environment will be from one of four *object types*: *points, lines, surfaces,* and *solids*.

Initially there is just one *object* in the 3D environment, it is a *line* which is (essentially just like the Z axes) a straight path extending to infinity in both directions, and exists perpendicular to, and at the intersection of, the X and Y axes of the 3D environment.

Points are input as three individual numeric values, each of these values informing the *point's* position along one of the three axes of the 3D environment, its X, Y and Z coordinates. Any number of *points* may be placed in the 3D environment.

Operations: These *point/s* and *line*, HRTA *primary objects,* may be used as input to inform one of five processes, *operations*, which can produce new *points*, *lines*, *surfaces,* and/or *solids*, HRTA *secondary objects*. These *secondary objects* may then be used as input to inform new iterations of the *operations* to produce new *secondary objects*. (One way a *surface* may be produced is as the result of a *trail operation* with a *line* as its *vehicle object*. One way a *solid* may be produced is as the result of a *trail operation* with a *surface* as its *vehicle object*. See *trail*, ahead.)

A connection is maintained between each *secondary object* and the original *object/s*, HRTA *contributing object/s*, which informed the *operation* by which it was created, restricting the manipulation of a *secondary object* to be only possible by editing its *contributing objects*. (Example: The X, Y and Z values of a *secondary point* may not be modified directly, but rather they may only be changed by manipulating its *contributing object/s*.)

Measurements: *Lines*, *surfaces,* and *solids* may either extend to infinity in every direction, HRTA *infinite*, only certain directions, HRTA *partially-finite*, or not extend to infinity in any direction, HRTA *finite*. A *measurement* value becomes available for all *finite objects*.

Line: A *length* value becomes available for any *finite line*.

Surface: An *area* value becomes available for any *finite surface*.
Solid: A *volume* value becomes available for any *finite solid*.

A *partially-finite* or *finite object* will have either one or some combination of *points*, *lines,* and/or *surfaces* positioned at the location/s where it ceases to extend to infinity, HRTA *boundary objects*.

3D Operations

Bend Space

Bending is a method by which *object/s* in a first category, HRTA *target objects*, are deformed in response to the changing location and/or form of *object/s* in a second category, HRTA *control objects.* In short*: Unbent Ctrl obj/s + Bent Ctrl obj/s + Unbent Target obj/s = Bent Target obj/s* (See fig. 4)

- - - - -

Space: To illustrate *bending*, envision that *space* (the entire virtual 3D environment) consists of a malleable, metaphysical matter which can vary in density (being expanded in places and/or compressed in others) as well as somehow be folded into itself so multiple portions of itself may all occupy the same shared location of the original (non folded) 3D environment in which they exists.

Target Object: The *target object* is the *object* you wish to deform.

Control Objects: *Control objects* are the *objects* which inform the deformation. *Control objects* consist of any number of pairs of *objects.* One from each pair is assigned to one of two categories, *unbent* and *bent.*

Bend: Envision that the *target object* and all the *control objects* assigned to the *unbent* category are fused to become one with the *space* they occupy, so they now all act as one and whatever deformation occurs to the *space* will be reflected in the form and/or location of the fused *objects*. Now imagine that the *space* morphs from its original state, *unbent state*, where it is of one homogenous consistency throughout, to a second state, *bent state*, where it is distorted in the flowing manner. The fused *control objects* from the *unbent* category are reshaped and relocated to now share the exact position of their counterparts assigned to the *bent* category. (When a *control object* is paired with itself the *space* it occupies is secured so as to remain unchanged by the *bend operation*.)

A zero to one hundred *efficacy* value can be set for each *control object* individually. This value dictates how much influence that particular *control object* has in the *bend operation*.

A variable is available which can change the intensity of the curvature individually for each *secondary object* produced by the *bend operation* (Example: If you rotate by 45 degrees one end of a straight *finite line* around an axis which is both perpendicular to that *line,* and intersects that end of the *line*, and also rotate by 45 degrees but in the opposite direction the other end of that *line* around an axis which is parallel to the fist axis, and intersects that second end of the line, this variable can change the curvature of that *line* from flatter to more extreme, or vise versa. See fig. 1000) More options may be introduced in the future which in different ways manipulate (as does this variable) the particular mathematics employed by the *bend operation* to *bend space*, changing the particular form of the *secondary object/s* produced by the *bend operation*.

Secondary objects produced by the *bend operation* will not necessarily be of the same *object type* as the *input target objects* which informed their creation. (Example: An *unbent target object* of *type line*, placed between two *unbent control surfaces* which share one *surface* as their *bent* counterpart, can collapse into a *target object* of *type point.*)

Analogy of the Bend Operation

The following is an analogy to help visualize the *bend operation* and the components involved. (Note: this analogy does not have a correlating parallel to every element of the actual *bend operation*, but is instead just intended to illustrate the general function of the *bend operation*. For instance, 'self intersection' is not an element represented in this analogy.)

Components (for the upcoming text):

- Foam = *space*
- Flat layer of glue = *input target object*
- Deformed layer of glue = *output target object*
- Deactivated magnets = *unbent control objects*
- Activated magnets = *bent control objects*

Imagine a piece of foam with a number of little magnets embedded at random locations and orientations within it. These magnets can be controlled remotely to either be magnetized or demagnetized. (**Magnetized:** where they act as magnets by attracting and/or repelling each other, or **demagnetized:** where they do not behave as magnets and are indifferent to each other's position). When the magnets are magnetized, the piece of foam becomes slightly deformed in a manner which reflects the attraction and/or repulsion of the magnets.

Now imagine the foam is constructed by gluing together two flat faces of two individual pieces of foam.

Envision the deformation occurring to the flat layer of glue sandwiched between its two pieces of foam, when the magnets become magnetized. This deformation is analogous to the *bending* of the *bend operation*. The layer of glue is *bent* in response to the changing position of the magnets.

Input: Three groups of one or more *objects*, *unbent control object/s, bent control object/s* and *unbent target object/s*. Each *object* in each of the two *control* groups must be paired with at least one *object* in the other *control* group.
Output: For each *input target object* in a *bend operation* a new *secondary object* is produced.

Lever (Enhancement to the Bend operation)

A *lever* is a tool which works within the *bend operation* to identify (acting as a *target object*) or specify (acting as a *control object*) the density and orientation of the *bent space* in immediate contact with any particular *object*, HRTA *attached object*, by utilizing an additional *object*, HRTA *detached object* (See fig. 6, 7, and 8).

(The function of the *lever tool* is apparent only when *space* is also being influenced by other *control object/s* present in the *bend operation* along with the *lever*.)

- - - - -

Independently of each other, the *attached* and *detached objects* of a *lever* may each act as either a *target* or *control object*. When the *detached object* is a *target object* the *lever* is considered a *target lever,* and when the *detached object* is a *control object* the *lever* is considered a *control lever*.

***Lever* with a *target object* as its *detached object*:** Although no actual *line object* is involved with a *lever* whose *attached* and *detached objects* are both *points*, it may be easiest to understand the function of this *lever* by envisioning a “line” spanning between these two points. This “line” is not *bent* by the *bend operation*, as would be an ordinary *line target object*, but instead behaves as an unbendable straight "line" that is fused to the *bending space* at only one of its ends, the *attached* end, leaving the other end, the *detached* end, free to be repositioned by the *bend operation.*

Relative to the *unbent state*, the direction away from the *attached point,* to which the “line” is pointing in the *bent state*, indicates if and how the *space* in immediate contact with that *attached point* has revolved around it due to the *bend operation*. Relative to the *unbent state*, the length of the “line” in the *bent state* indicates if and by how much the *space* in immediate contact with the *attached point* has expanded or contracted due to the *bend operation*.

***Lever* with a *control object* as its *detached object*:** For the *lever* to act instead as a *control object*, an additional *point* must be *input* which identifies the desired location of the *detached* end of the “line” in the *bent state*. The *space* in immediate contact with the *attached point* is then orientated and scaled by the *bend operation* in a manner which repositions the *detached* end to be located at the *point* assigned.

A *lever* with an *object* other than a *point* as its *attached* end, behaves as theoretically would an infinite number of the “lines” with the *attached* end of each “line” positioned on one of every conceivable location on the *attached object*. A *lever* with an *object* other than a *point* as its *detached* end, behaves as theoretically would an infinite number of the “lines” with the *detached* end of each “line” positioned on one of every conceivable location on the *detached object*. For a

lever with *objects* other than *points* at both its ends, each of the infinite number of “lines” will be positioned on the end *objects*, at correlating locations.

Input: To make any *target* or *control object* behave as the *attached object* of a *target lever*, input another *object* to serve as its *detached object*. To make any *target* or *control objects* behave as the *attached object* of a *control lever*, input one *object* to serve as its *detached object* in the *unbent* position and a second *object* to serve as its *detached object* in the *bent* position.
Output: For every *target detached object* a new *secondary object* is produced.

Spin (Enhancement to the Bend operation)

(As is the case with *levers*, The function of the *spin tool* is apparent only when *space* is also being influenced by other *control object/s* present in the *bend operation* along with the *spin*.)

A *spin* is a tool which works within the *bend operation* to allow you to rotate a *control lever* whose *objects* are *points,* around its own axis (the "line" defined by its two *points*), but instead of how *space* typically may be folded into itself as it is resolved between neighboring *control objects*, around the *attached point* of a *spin*, *space* spirals, stretching into ever thinner, accumulating, concentric, non-converging layers. In other words, whereas with other *control objects* you may revolve *space* around any particular spot only up to 180 degrees before you start approaching zero again, with a *spin* the rotation can continue indefinitely to any number of cycles, without ever coming back to zero (See fig. 10).

Input: Assign a numeric value defining the degree of rotation to any *control lever* with *points* as its end *objects*.

Intersect

Objects produced by the *intersect operation* will fall under either one of the following two categories: *junctions* and *segments* (See fig. 12, 13, 14, 15, 16, 17, and 18).

Junctions: *Junctions* are new *objects* created at the location where either two or more *objects* come into contact with each other, where an individual *object* is creased to the extent that there is no longer tangency at that location, or at a location where an *object* comes into contact with another part of itself. (Example: A *point* may be created at the location where three *surfaces* all come into contact with each other.)

Segments: When a first *object* is divided into multiple distinct portions by either another object/s, or some other portion/s of itself, passing through it, that first *object* produces an individual *secondary object* for each of those distinct portions. These new *secondary objects* are HRTA *segments*. (Example: A *line* which is in contact with a point somewhere along its length will produce two *line segments*.)

Input: Input any number of *objects*.
Output: For each input object information is provided identifying the segments it results in. Along with each pair of neighboring segments the particular junction object which divides them is identified. When a resulting junction object is itself segmented within the same intersect operating in which it itself was created, an unsegmented version of itself also becomes available.

Trail

Trails are *objects* whose form is a representation of the continuous, cumulative location a traveling *object,* HRTA *marking object*, would occupy throughout its entire journey if the time of that journey was collapsed into a single moment.

- - - - -

The particular journey of the *marking object* is defined by having an *object* which influences the form of the *marking object*, HRTA *vehicle object*, transverse the extent of yet another *object*, HRTA *route object*. A *route object type* must be one dimension higher than that of the *vehicle*. (*Line* is one dimension higher than *point*, *surface* is one dimension higher than *line,* and *solid* is one dimension higher than *surface*. Only *route objects* that when traversed by the *vehicle object* will in turn cause the *marker object* to move as well, will yield results. Example: a *bent control point* (a *control point* assigned to the *bent state* of a *bend operation*) that were it to transverse the extent of a given *line* would cause a particular *target point* to move, can be used as a *vehicle object* with the *line* as its *route*, and the *target* as the *marker*. See fig. 20, 21, and 22)

A single *trail operation* may have multiple *marking, vehicle,* and/or *route objects*. Any number of *vehicle objects* may share the same *route object.*

Input: At least one pair of *objects* from *types* of consecutive dimensions (0&1, 1&2, or 2&3) to serve as the *vehicle* and *route*, and at least one *object* influenced by the *vehicle object* to serve as *marker*.
Output: For each *input marker object* in a *trail operation,* a new *secondary object* is produced.

Parallel/perpendicular

The *parallel/perpendicular operation* identifies the location/s on a curved *line* or *surface*, HRTA *specimen object*, where the tangency of the *object* becomes either *parallel* or *perpendicular* to either any assigned axis (in any conceivable direction, not just the three axes of 3D environment) HRTA *linear axis*, or to any conceivable axis pointing away from an assigned *point,* HRTA *point axis* (See fig. 24, 25, 26, and 27).

Input: Enter either a single *point* which will behave as a *point axis*, or select a location on any *line object* to inform the direction of a *linear axis.* Enter a *line* and/or *surface object* to behave as a *specimen object.*

Output: Resulting *objects* will fall into one of two categories, *parallel* and *perpendicular*.
Line: Results on a *line specimen* will be of *object type points*.
Surface: Results on a *surface specimen* will be of either *object type points* and/or *lines*.

Radius

Radius is a method by which to identify the *radius* of the curvature at any specified location on a curved *line* (See fig. 29).

The *radius operation* produces a *point* located at what would be the center of a theoretical circle drawn to be in tangent with the specified location of the specified *line*, HRTA *radius line*.

(If no curvature exists on the *radius line* at the specified location, no *secondary point* is produced.)

Input: A *line object*, and a numeric value indicating the desired coordinates on that *line.* **Output:** A single *point object.*

Split

The *intersect* and *parallel/perpendicular operations* may produce *lines* which run in never-ending loops and therefore cannot be used when a beginning and end are required.

To assign a beginning and end to a loop so it may serve as a *finite line*, it may be split at a location selected along its path, and have one side of the split assigned as the direction in which the value of its coordinates runs in the positive direction.

Utilities

Utilities are elements of the software which aren't directly involved in the formation of 3D geometry.

Material

Material is a collection of properties which dictate the appearance of objects. Objects may appear with one of four main material categories: uniform, display, mask or gradient.

Uniform: U*niform* is a collection of characteristics applied consistently throughout an *object*.

- **Color:** A value identifying a position along the spectrum of visible light.
- **Saturation:** A value controlling the potency of that *color*.
- **Shade:** A value controlling the brightness.
- **Opacity:** A value controlling the transparency of an *object*.
- **Luminous:** A value controlling the amount of light emitted from an *object*.

(Unlike with the *uniform* feature which enables you to view an *object* itself, *display*, *mask* and *gradient* cause the *object* to work as a portal through which to view other *objects*.)

Mask: A *mask* allows you to view in place of an *object*, HRTA *mask object*, the portions of any other *object/s*, sharing with the *mask object* a sightline from the *display* vantage capturing them.

Gradient: A *gradient* allows an object to appear as a fluid blending of the appearances of any number of assigned objects. (To envision how a *gradient* looks, imagine one red and one blue candy suspended in jello. Now imagine those candies were in there for a while and started to stain the jello around them to the point where every portion of the jello is either red, blue or some combination of the two depending on that particular portion of jello's proximity to each of the candies.)

Although *points* and *lines* are on their own not visible (since they are infinitely thin), material properties may be assigned to them so they may participate in informing the appearance of a *gradient*.

Uniform material properties may be extracted from any selected location of a *gradient*.

Portal Properties: (Properties shared by the *display*, *mask* and *gradient* features.)

- **Shadow:** you may specify that light from any specific source (*object* with a positive *luminous* value) travels without obstruction through any specified object to hit any other specified object. In other words, for each light source you may specify which *object/s* can block its light (cast shadows) from reaching which other *object/s,* regardless of all their relative positions.
- **Realistic:** A *display* may either appear rendered, simulating the natural effects of light, or appear flat, where the silhouettes of shadows appear as a crisp edge with no diffusion.

(The following are material properties which either apply only to *surface objects* or only to *solid objects*.)

Surface:

Obscure: When two portions of *surface objects* occupy the same location, they may either blend together appearing as a combination of their assigned material properties, or one may be selected to obscure the other.

Double sided: Material properties may be assigned to either one or both faces of a *surface*. When assigned to both its sides the appearance on the back is an inversion of what appears on the front. When assigned to just a single face, the opposing face appears fully transparent.

Reflect: A value controlling the intensity with which the reflections of neighboring *objects* appear on a *surface*.

2D Display: The view of any 3D environment may be shown on any *surface object/s* and/or 2D *output device/s* (Example: physical monitor), from the vantage of the center of any selected *surface object,* looking away from that *surface's* front face, and rotated around its line-of-sight at an angle matching the coordinates of that *surface* (See fig. 31).

Lens:

- **Perspective:** Allows you to view the *display* orthographically or with the distortion of perspective.
- **Shutter speed:** Allows you to mimic the effects any change to a shutter speed would have on the way a *display* appears.
- **Aperture:** Allows you to mimic the effects any change to the size and shape of an aperture would have on the way a *display* appears.

Solid:

Refract: a value controlling the amount light is refracted as it travels through a *solid* with less than 100% opacity.

3D Display: The view of any 3D environment may be shown in any *solid object/s* and/or 3D *output device/s* (Examples: holographic display, virtual/augmented reality set).

Multiple *2D/3D displays* may be assigned to a single *output device*. Any two *2D/3D displays* assigned to the same *output device*, may either blend together or one may be selected to *obscure* the other.

Object/s may be assigned different material properties in each *2D/3D display* they appear.

Input/output devices

For any connected *input/output device*, specific information becomes available. (Examples: For buttons on a keyboard or mouse two pieces of information are typically available, a name of the button, and a yes/no value indicating whether the button is being held down or not. In addition to yes/no button values, a mouse typically also has available two values representing movement along X and Y axes. For each finger on a touchscreen or trackpad three values are typically available, two indicating its position along the device's X and Y axes, and a third value indicating the amount of pressure being applied in that location. X and Y values are available for both the physical dimensions and pixel count of a physical monitor.)

Sharing

A user may grant permissions to any other user/s and/or group/s of users, giving them access to any specified element/s in this software.

Custom properties

Assign to any particular kind of element in the software, any number of new properties, so along with the values already required for that kind of element, values for these new properties are also stored. (Example: In addition to the X, Y, and Z properties of a *point object* a fourth and fifth property, for instance, may be added, one named ‘Politics’ and the other ‘Religion’. Values may then be assigned to those properties. These new properties may now be used to sort, filter, and/or perform any other possible task/s with *point objects* to which these new properties were assigned.)

Activity log

A record, HRTA *activity log*, is kept of everything that happens in the software. Any value can then be returned to at a later date for modification, causing any elements influenced by that value to be automatically updated accordingly.

Audio

Recorded sound is stored as a *finite* 3D *line object* whose shape represents the recording's change in amplitude over time. (This *line object* may be utilized in the 3D environment as would be any other *line object*.) For any sound received from a microphone, a numeric value is available which indicates the current amplitude of that sound. An audible sound may be produced whose amplitude is dictated by any assigned numeric value.

Token

A *token* is a predefined criteria. (Example: a user can create a *token* which requires three *point objects*. A tool may now be created which produces a triangle from any element/s that satisfy this *token*.) The definition of any *tokens* is able to be permanently secured in the blockchain. You may permanently secure any element/s which satisfy a particular *token*, so the only changes allowed to be made to those element/s are ones which satisfy the *tokens* criteria. Any *token* may be made to require other particular *token/s* as part of its criteria.

Default toggle

A method is always available by which to access a preselected, default interface.

Community

A web (internet) based way for users to share their digital creations. The relative success of different creations sharing similar objectives (perhaps created by unacquainted parties) may be computationally evaluated and determined. Users can also vote on which is their favorite. Users can elect to have their default interface automatically updated to the latest version posted by any particular user they choose.

Property Values

Lock value: The value of any particular property is able to be individually locked so that it remains unchanged. (Example: the *length measurement* value of a *finite line* is able to be locked so that the position of the boundary *point objects* establishing the *line's* ends may only be changed in ways which allow the *length measurement* value to remain unchanged.)

Conditional value locking: Conditions can be set so that when they are met certain values are either locked or unlocked.

Maintain secondary *object*: Similar to how a value can be locked, the existence of any particular *secondary object* may be secured as well. (Example: the existence of a particular *junction point* may be secured so that the geometry of its *contributing objects* may change only in ways that allow that resulting *point* to remain in existence.)

Resolve: In an instance where the value of a particular property can only be changed if the value of some other (not locked) property is changed as well, the software can detect this and automatically change that second value by the appropriate amount.

Relationships: Specific relationships can be assigned between the values of any specified properties. (Example: the value of one property can be constrained to only equal a certain percentage of the value of some other property.)

Scripting

Nesting: A *script*, HRTA *host script*, can be made to execute any number of other *scripts*, HRTA *embedded scripts*. The *host Script* may send any value/s for processing to any *embedded script/s*.

Autocorrect scripting (See fig. 32)

To form an executable script from an ordinary text the software first sorts all the words/phrases into their appropriate categories (see examples of word/phrase categories below). These categories have an order of priority.

To function, a particular word/phrase may require any number of other particular word/phrase/s. (Example: the word ‘plus’ requires two supporting words/phrases which are able to be added to each other.)

Multiple words/phrases may share the same function. (Example: plus, and, add, combine…)

For each instance where a specific collocation of words/phrases can be rationally arranged in more than one way, or where some conflict is present, the user is offered a selection of possible solutions to choose from.

Potential Autocorrect Scripting Word/Phrase Categories

Elements: Names of specific elements within the software. Example: *point*, *intersect*, *token*.

Scripts: Names of other *scripts* or variables within a *script*. Example: Bob's circle, Midterm Project, Birthday.

Navigational: Words determining the tasks to be performed and the sequence in which they occur. Example: If, then, go to.

Mediators: Words informing how two portions of a script relate to each other. Example: between, following, above.

Logical Operators & Numbers: Symbols or words informing how two portions of a script are evaluated relative to one another, or numeric values. Example: The plus sign (+), thirty four, without.

External Data: Piece of information from sources unrelated to our software. Example: Time, weather, special of the day posted on the website of your favorite spot.