12-4-2014

Assembling 3D Objects with Artificial Spatial Intelligence

Eugene Koon

Rochester Institute of Technology

Department of Computer Science

B. Thomas Golisano College of Computer and Information Sciences

December 4th, 2014

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in Computer Science

Assembling 3D Objects with Artificial Spatial Intelligence

By Eugene Koon

Chair:	Roger Gaborski
Reader:	Reynold Bailey
Observer:	Leon Reznik

Committee Approval

Roger Gaborski — Date

Chairperson

Reynold Bailey — Date

Reader

Leon Reznik — Date

Observer

Abstract

The focus of this thesis is to provide an artificial intelligence (AI) system that can develop spatial intelligence. A MATLAB application of a genetic algorithm AI system has been implemented. The AI system will incorporate three dimensional (3D) objects that can learn to maneuver in 3D space so that they may be assembled with each other. As an example, two rectangles with holes and a nut will maneuver itself onto a screw. The performance of the AI system will then be recorded as video results.

Table of Content

Introduction

One may wonder how smart are artificial intelligence (AI) systems? Can a standard IQ test grade their intelligence level? On July 2013, MIT's AI system called ConceptNet4 [Diochnos 4] was placed under an IQ test that was administered by Wechsler Preschool and Primary Scale of Intelligence [Arnold 1][Ghose 9]. The results had shown the AI system to have an intelligence level of a four year old child. It had scored exceptionally well under the verbal section, but poorly under the comprehension abilities (the answering of "why" questions) and common sense knowledge ("don't touch the hot stove") sections.

Robert Sloan, a professor and head of the computer science department at University of Illinois in Chicago commented, "As babies, we crawled around and yanked on things and learned that things fall. We yanked on other things and learned that dogs and cats don't appreciate having their tails pulled" [Arnold 1][Ghose 9]. These behavioral learning practices, as well as others, allow human beings to learn. But can an AI system also learn by yanking and pulling, or by generally interacting with the three dimensional (3D) world to become smarter?

Stephen E. Arnold, Managing Director of Arnold Information Technology (ArnoldIT) states, "Common sense knowledge in ConceptNet encompasses the spatial, physical, social, temporal and psychological aspects of everyday life" [Arnold 4]. Improving these aspects could lead to improvements to an AI's common sense ability. If an AI system is provided with spatial intelligence, it may provide the methodology for the AI system to further develop its own common sense knowledge.

This thesis is focused on creating an artificial intelligence (AI) system which develops spatial intelligence. One of the ways human beings learn about spatial awareness is by interacting with three dimensional (3D) objects, such as toys. An example of a toy that improves on spatial intelligence is a Toddler's Toy Box shown below under figure 1.

Figure 1: A Toddler's Toy Box Helps Toddlers Improve Their Spatial Intelligence[1]

A Toddler's Toy Box is made up of several small shaped blocks and a large box. On the surface of the large box are several shaped holes that correspond to the several small shaped blocks. The goal of the toy is to have a toddler place all the small blocks inside the box. If the toddler fails to place a block inside the box on the first attempt, he/she would continuously try again. These continuous behavioral attempts seem similar to how a genetic algorithm (GA) would continuously create mating pools of several options and select one of the options as the next attempt. If and when the toddler decides to try another side of the box, this again seems similar to how the genetic algorithm may mutate its options.

As the toddler continues to interact with the small blocks, he/she learns to dynamically manipulate the objects in specific ways so that it may be placed inside the large box quicker. This act of dynamically learning is similar to how a weight schema would enforce stronger trends. This is essential because different 3D objects are affected by different trends. For example, rotating a spherical block will not change how the object would interact with the box. However, rotating a triangular block will change how the object would interact with the toy box.

This concept of playing with the toy box is the inspiration behind creating a weighted GA system to

[1]Guangzhou Cyber Technology Co., Ltd "Colorful Wooden Toy Children Intellect Box 13 Different Shapes of Holes Matching Game " 2013. 10 Nov. 2014 <http://www.aliexpress.com/store/product/Colorful-Wooden-Toy-Children-Intellect-Box-13-Different-Shapes-of-Holes-Matching-Game-Free-Shipping/916337_728585784.html>

develop spatial intelligence. It seems possible to create an AI system that can learn to assemble the 3D objects in 3D space with each other. It may also be possible in the future that similar systems may automatically assemble objects together with just a blueprint. For that reason, instead of using the elementary Toddler's Toy Box as an example, in this thesis I will use MATLAB to assemble two rectangles with holes and a nut onto a screw. Shown below under figure 2 are four images of objects used for this thesis.

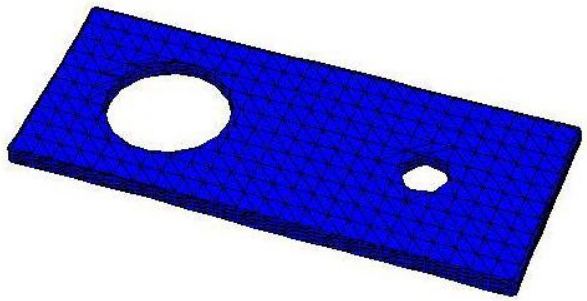
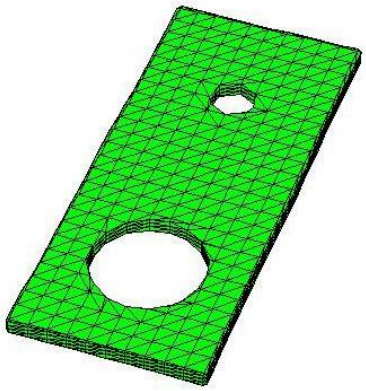
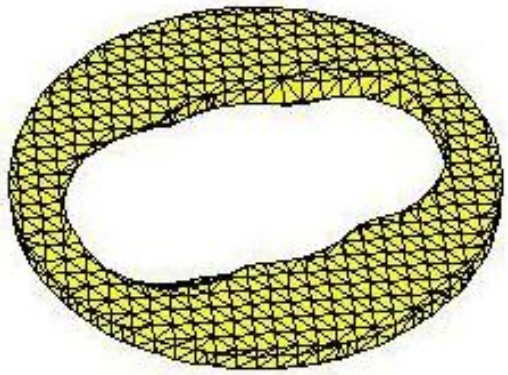
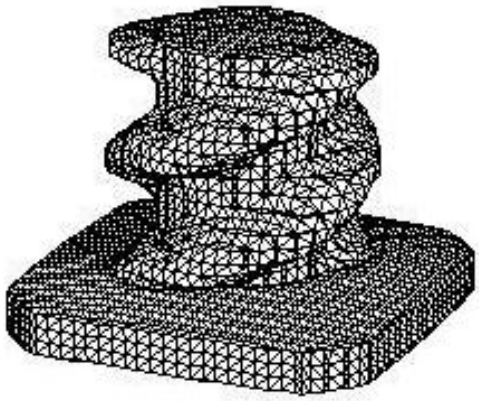
	
Rectangle 1	Rectangle 2
	
Nut	Screw

Figure 2: Images Of 3D Objects To Be Assembled Together

Overview

A weighted GA AI system was implemented to control and manipulate 3D shapes in such a way that they can be assembled together. To demonstrate the performance of the system, several videos are provided in the results section of this thesis. The following objectives have been applied to the system:

1. Shapes will have the ability to move in the X, Y, and Z direction.
2. Shapes will have the ability rotate using axis angle rotation.
3. Shapes are allowed to be mutated to new locations.
4. Shapes cannot collide or share any point in 3D space with each other.
5. In a given scenario, two objects will be joined together at a time. These objects will be labeled as either an active or a static object. Active objects will be able to maneuver in 3D space while static objects will not.
6. Shapes will be considered combined when the active objects have moved into a specified goal location located near the static object.
7. MATLAB is the coding language that will be used

Background

Having a computer render solid 3D objects is both difficult and a huge computational workload. It is simpler to represent 3D objects as a mesh like net of vertices points surrounding the object's surface. This can be done with Constructive Solid Geometry (CSG) objects. There have been some previous CSG projects with MATLAB by Bruce Land from Cornell University [Land 7]. However, there are two constraints when dealing with CSG objects. CSG objects have rigid motions when translating and rotating the object and collision detection algorithms have to be generalized to accurately detect collisions. "To improve the resolution process, Ambler and Popplestone developed techniques for separating the rotational and translational parts of rigid motions" [Rossignac 14]. This is done to "eliminate linear occurring variables" [Rossignac 14] that are exposed due to translating and rotating the object to a desired location and orientation with just one step. Separating the process of motion provides a simple solution without having a complicated algorithm. The process of translating an object is very simple. All the vertices points are shifted by the same offset. However, the process of rotating an object is more complicated.

To rotate a given object, there are four methods of implementing rotations that have been widely used. See figure 3 for an illustration of the four types. The first method is known as axis angle rotation which involves rotating the object around a provided axis. The second method is known as Euler angle rotation. This involves three angles for three rotations to be performed on each axis of the 3D space. The third method is known as matrix rotation, where a matrix of trigonometry properties is used to represent the rotation. The last method is known as quaternion rotations and involves four dimensions to be used to effectively rotate an object. This incorporates one real and three imaginary dimensions to rotate an object by four angles. All methods can be converted from one method to another, but they do not share the same qualities.

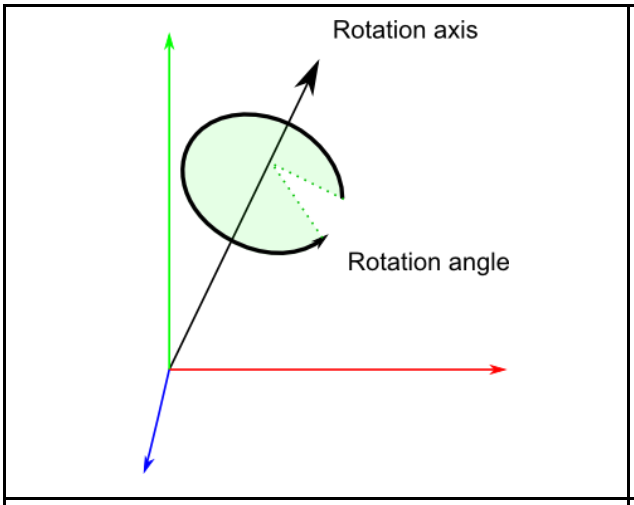
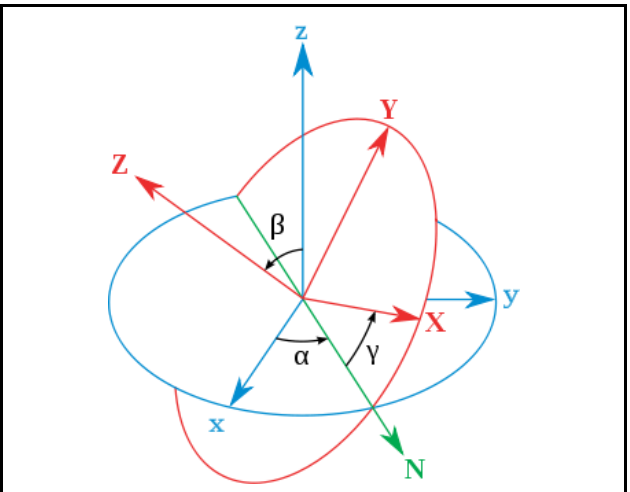
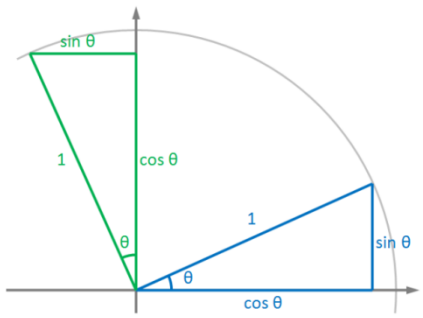
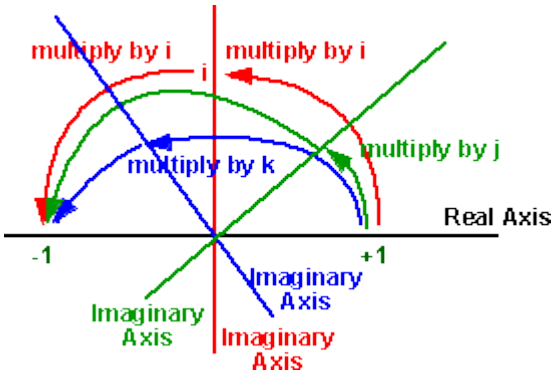
	
Rotate Via Axis Angle Rotation[2] Objects Rotate Around a Rotation Axis	Rotate Via Euler Angle Rotation[3] Objects Rotate About Angles of the Axes
	
Rotate Via Matrix Rotation[4] Objects Rotate With Matrix Representing Trigonometry Properties	Rotate Via Quaternion Rotation[5] Objects Rotate Based On One Real Axis And Three Other Imaginary Axes

Figure 3: Four Types Of Rotations

[2] "Tutorial 17 : Rotations | opengl-tutorial.org." 16 Nov. 2014 <http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/>
[3] "Euler angles - Wikipedia, the free encyclopedia." 2004. 16 Nov. 2014 <http://en.wikipedia.org/wiki/Euler_angles>
[4] "Rotations and Infinitesimal Generators | Algorithms and ..." 2011. 16 Nov. 2014 <http://reedbeta.wordpress.com/2011/09/18/rotations-and-infinitesimal-generators/>
[5] "Maths - Quaternions - Martin Baker - EuclideanSpace." 2003. 16 Nov. 2014 <http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/>

One major rotation problem is called the Gimbal Lock, see figure 4 for an illustration. This is only apparent under the Euler rotation, in which the object cannot rotate along one axis because one degree of freedom has been lost while rotation was performed by the other two axes. This can be avoided by using the other rotation methods mentioned. However in this thesis, Gimbal Lock will not be a problem because Euler rotations are only performed to randomize the CSG objects. After the objects have been randomized, objects thereafter will be rotated via the axis angle of rotation method.

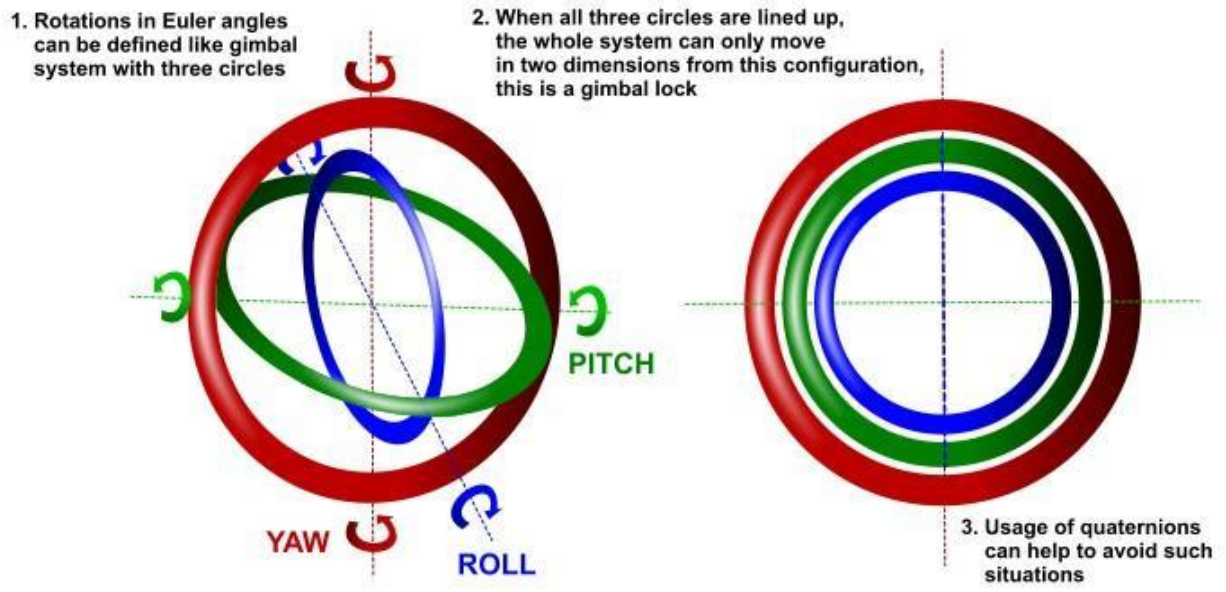


Figure 4: Illustration of Gimbal Locking[6]

When translating or rotating multiple objects in the same 3D space, objects may undesirably collide. To prevent this, a collision detection algorithm is needed. A typical collision detection algorithm needs to be fast yet accurate. However, the most computational scenario of algorithm is when the objects have not yet collided. This is due to the fact that all the vertices points of the objects are compared to check if they are equal. To resolve this scenario, many implemented collision detection algorithms have used a preliminary bounds fast rejection test [Hadap 6]. Each object is generalized with a particular simple shape, such as a rectangle, and if the simple shapes do not overlap a collision has not occurred. See figure 5 for an illustration of the simple shape bounds. This logic is comparatively faster than verifying all vertices points. If the simple shapes do overlap, further testing is required to verify if the objects have collided. One example is to use multiple bounding shapes to represent each object. A second example is

[6] Fauvel, Cyrille. "Avoid Gimbal Lock for Rotation/Direction Maya Manipulators" Aug 10, 2012. <http://around-the-corner.typepad.com/.a/6a0163057a21c8970d017743e0b713970d-pi>

to use multiple polygons formed from the mesh like vertices points for each object and to check for crossing of polygon sides. Another example is to check if the object's centers are too close to each other using simple close proximity tests. There are many other methods, each with their own benefits and disadvantages depending on the type of shapes.

Figure 5: Illustration Of Primitive Bounds [Hadap 6]

In this thesis, a GA AI implementation is needed. It is one of the most powerful AI algorithms. Inspired by Darwinian principles of evolution, John Holland first introduced the adaptive algorithm that used probabilistic ideals to solve complicated problems in the 1960s [Holland 10]. The concept of the algorithm was created to mimic how organisms would evolve over time.

An organism's genes are comprised of various chromosomes that give features to that organism. When the organism reproduces, the next generation offsprings are formed from a pool of genes. "But occasionally, a change may dictate a completely new way in the evolution of those organisms" [Dinis 3]. Thus occasionally, altered offsprings would be generated due to crossover or mutation. Crossover occurs when two genes swap half their chromosome to form two newly arranged genes. Mutation occurs when one of the chromosomes of a gene changes to form a whole new combination for a gene. The offspring, normal or altered, must then survive or be eliminated due to the presence of unfavorable genes for a given environment. This natural selection will eventually force the organisms to continue carrying only the favorable genes from generation to generation for various environments.

This concept of adapting from generation to generation to create the perfect organism is what drives Holland's algorithm to find the ultimate solution to a complex problem. The algorithm initially creates random values for the first generation of solutions. During each following generation, new combinations of genes will form the next generation of solutions. Occasionally, genes are altered to explore more possible solutions via crossover and/or mutation. Each solution is scored and only the best fit solutions are kept for the next generation's mating pool. This generation to generation natural selection will continue until the ultimate solution has been found as shown under figure 6. In addition, since the

solutions are constantly rescored during each generation, this algorithm can also solve dynamic problems and provide adaptive solutions.

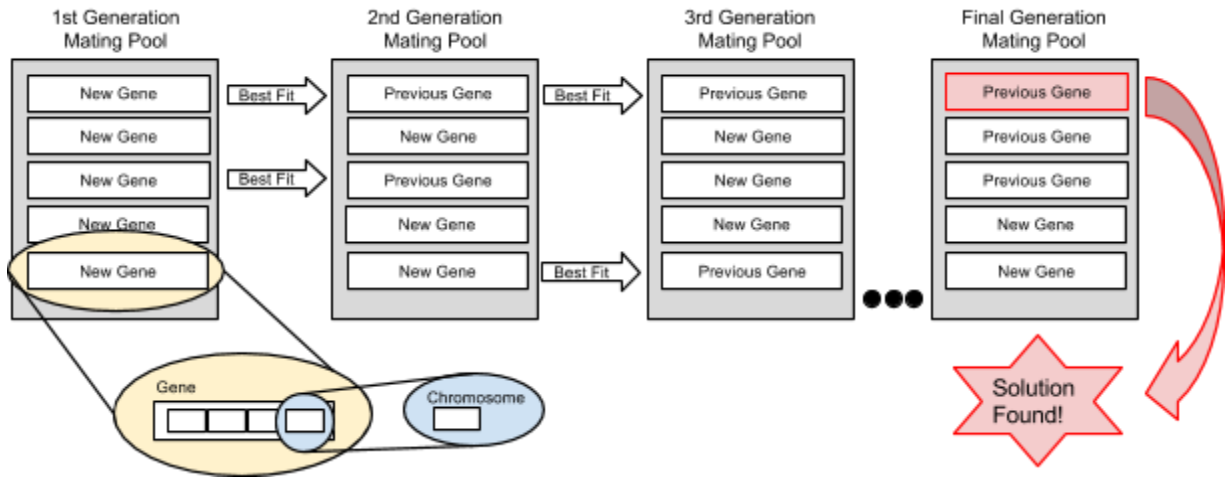


Figure 6: Illustration of Natural Selection From Generation To Generation

To further improve the GA to be more dynamic, a weighted schema can be used in conjunction with the GA. This can be done by implementing weights as chromosomes to each gene [Morgan 12]. From one generation to the next, natural selection of the best weight combinations will provide the ideal rate of change as smaller solutions. Eventually all the smaller solutions will lead to an adaptive solution for the dynamic problem.

Design

Description of the system

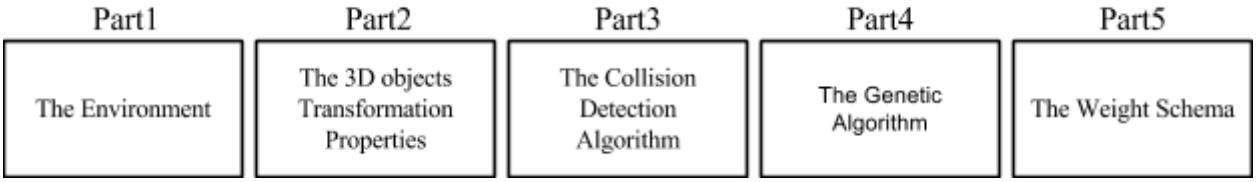


Figure 7: The Five Parts to the AI System

The designed AI system can be broken down into five parts as shown under figure 7. The first part focuses on creating the environment. This involves creating 3D objects within MATLAB in the form of Constructive Solid Geometry (CSG) objects. Two CSG objects are selected at a time to form a scenario. Each scenario has an active (changing) object and a static (non-changing) object. To evaluate any additional objects, the combined objects from the previous scenarios are set as the next scenario's static object while the additional object is set as the active object. See figure 8 for an example illustration.

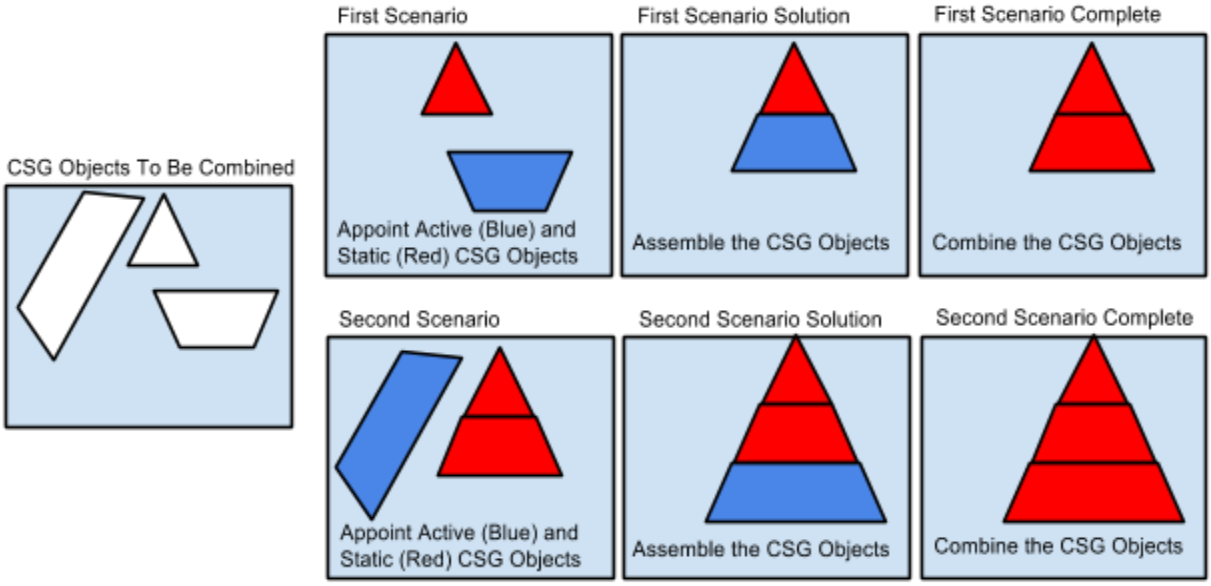


Figure 8: Illustration of Combining Multiple CSG Objects

The second part of the AI system will allow the active CSG object to maneuver in two ways. The active CSG object will be able to translate in 3D space, more specifically to have the property to be translated in the X, Y, and Z axis. The active CSG object will also be able to rotate in 3D space via either Euler rotations or axis angle rotations. Once the objects have been created from part one and their maneuvering properties have been implemented from part two, all the active CSG objects are randomly placed and rotated (via Euler rotation) for testing. The AI system will be evaluated based on how well the system can maneuver the randomly placed active CSG objects to their defined goal location and rotation orientation.

While moving and rotating the active CSG object in the same 3D space as the static object, it is important that the objects never overlap and take up the same space. For that reason, part three of the system incorporates a collision detection algorithm. All selected transformation options for maneuvering the active object require approval from the collision detection algorithm.

In the fourth part of the system, the heart of the AI system is comprised of a GA implementation. There are six processes to a typical GA implementation. These processes are listed below

I. Build a GA mating pool of genes
II. Perform any evolution changes (crossover or mutation)
III. Evaluate all genes from the mating pool
IV. Sort the mating pool
V. Select from the mating pool
VI. If the problem has not been solved, repeat the process with some strong performing genes

In this system, the GA mating pool is populated with a collection of possible new transformation options supplied by the weight schema. Each transformation option has a three percent chance of mutating the active CSG object to a new restarting location. Each transformation also has a score associated with how far the active CSG object is from its goal location. The GA sorts the mating pool by ascending scores and then randomly selects an option. If the selected option does not have a collision problem, the transformation option will be used to translate and rotate the active CSG object. If the active CSG object has a collision problem or has not been relocated to the goal location and rotation orientation, a new GA generation is needed. The mating pool is repopulated with the selected option, some old options with low scoring values, and some new options from the weight schema. The GA will continuously loop, until the active CSG object translates and rotates into the set goal location and orientation. See figure 9 for an illustration.

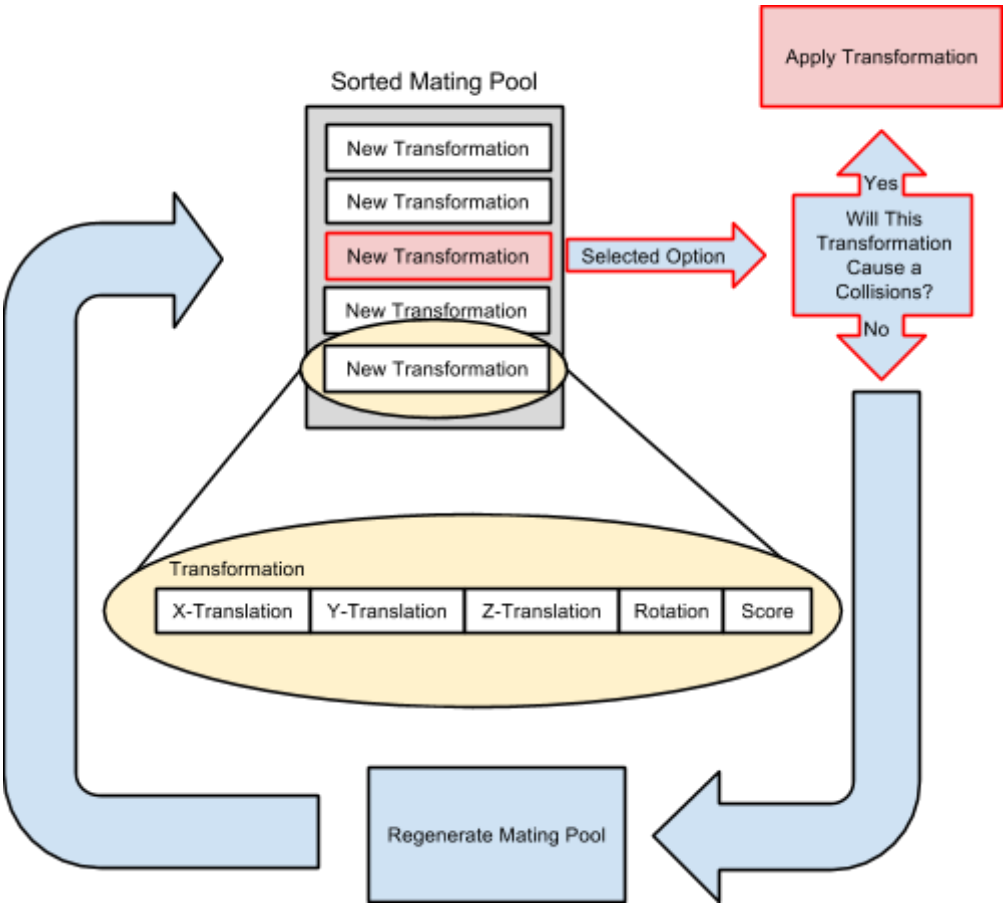


Figure 9: Illustration of Natural Selection From Generation To Generation

To maneuver the active CSG object, a weight schema was implemented as part five of the system. Because active CSG objects are constantly moving in this AI system, the problem to assemble the objects together slightly changes during each generation of the GA. Thus to solve the dynamic problem, an adaptive solutions must be found. This can be done via the weight schema. The weight schema is comprised of both weights, which acts like chromosomes to a gene, and a scoring system, which evaluates the gene. See figure 10 for an illustration of a transformation option/gene from the weight schema. There are four types of weights. The first three weights are determined by how far the active CSG object is to its goal location for each axis. The fourth weight represents how to rotate the active CSG object. The transformation option scoring system provides a score by taking only the sum of the first three weights. The higher the weight, the more the active CSG needs to be translated. Thus the AI system prefers lower scoring transformation options and passes these options from generations to generation. This will continue until the weights have adaptively solved the dynamic problem.

X Translation Weight	*Y Translation Weight*	*Z Translation Weight*	*Rotation Weight*	*Transformation Score*

Figure 10: A Gene From The Weight Schema

Creating 3D Objects with MATLAB

To represent 3D objects in MATLAB, constructive solid geometry (CSG) objects were created. CSG library files for MATLAB have been created by Bruce Land from Cornell University [Land 5]. CSG objects are formed from basic unit shapes of cylinders, spheres, rectangles cuboids, and torus. All these shapes can be scaled to any size, combined with each other, subtracted from each other, and used for their intersection. Having created the desired shape, the shape's surface is converted into a CSG mesh of vertices object that will be represented by some resolution value, matrices of vertices, matrices of faces (for shades of color), face color (the color of the object), and edge color. Shown in figure 11 is an example CSG object.

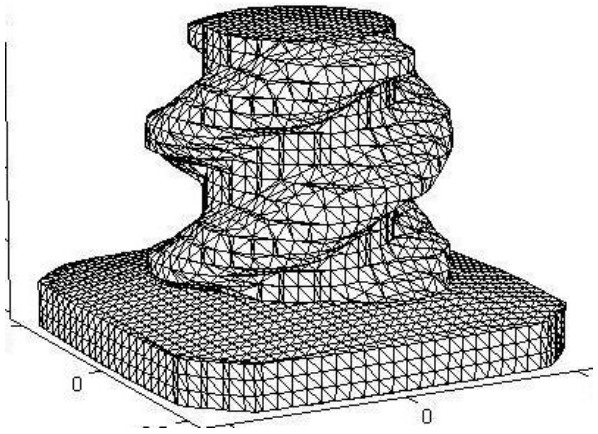


Figure 11: CSG Representation of a Screw

A CSG object can represent one or more objects. It can be made up of either a structure of an object or a cell of structures for multiple objects. Thus multiple objects can be combined into one CSG object. See figure 12 for possible CSG object values.

vertices: [15677x3 double] faces: [31346x3 double] facecolor: 'white' edgecolor: 'black'	vertices: [3671x3 double] faces: [7336x3 double] facecolor: 'yellow' edgecolor: 'black'	[1x1 struct] [1x1 struct]
Screw structure	*Nut structure*	*The combined screw and nut structures to form cell of structures*

Figure 12: Display Of How 3D Objects Are Represented

A property of the CSG structure objects are vertices. Vertices are formed from X, Y, and Z coordinates from all the points that make up the shape's surface. Thus the matrices of vertices are in the form of [m x 3], where *m* depends on the resolution of the object.

The resolution of the CSG object is very important and is scaled from one to fifty. A high resolution will allow the CSG object to have more vertices points to represent the shape's surface. The more vertice

points available, the more detailed the CSG object can be to represent the desired shape. See figure 13 for three examples of various resolutions. However having a high resolution will proportionally slow down the time required to handle processes such as translation, rotation, and collisions detection. Furthermore, having a low resolution will cause the CSG object to overly generalize the shape. This is especially apparent with shapes having sharp edges, like a screw with a resolution of five.

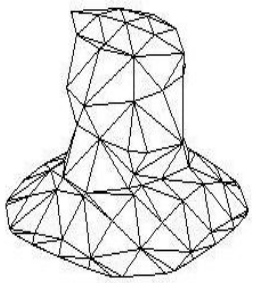
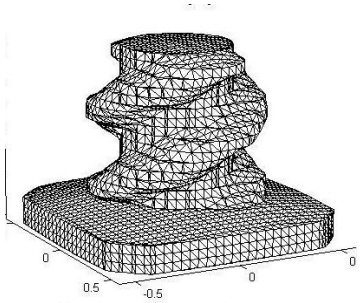
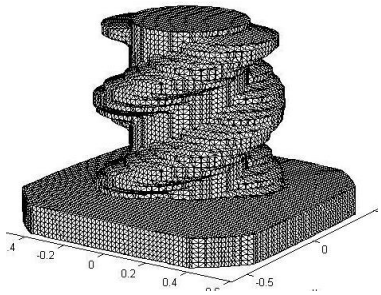
		
Resolution = 5	*Resolution = 25*	*Resolution = 50*

Figure 13: Display Of How 3D Objects Are Affected By Resolution

While the active CSG objects were created, there were two properties that were added to active CSG objects to assist the AI system in manipulating the objects in 3D space. These properties include a reference point and two extra unit vectors. These properties were then appended to the end of the vertices of the active CSG objects. As the active CSG's vertices are modified from a translation or rotation, these extra vertices are also modified in the same fashion.

The first additional property for the active CSG objects is composed from a vertex point called current goal reference point (CGRP). The CGRP is used to represent where in 3D space the current object is located and is kept dynamically updated. The CGRP is continuously compared with a predefined goal location point (GLP). The GLP is used to represent where the object wants to be located in the same 3D space as the static object. In the example with the "screw and nut", the CGRP is represented by the center of the active nut. In the example with the "screw and rectangles with holes", the CGRP is represented by the center of the desirable hole on the rectangle. For both examples the GLP is a point in 3D space that will cause the active object to translate onto the static screw.

The second additional property for active CSG objects is composed of two unit vectors. These vectors help the active object rotate via axis angle of rotation into a desired orientation. While the active CSG object was initially created and located at the desired location, the two orthogonal unit vectors are defined as *vector A[1,0,0]* and *vector B[0,0,1]*. See figure 14 for an illustration of the two unit vectors on two active CSG objects.

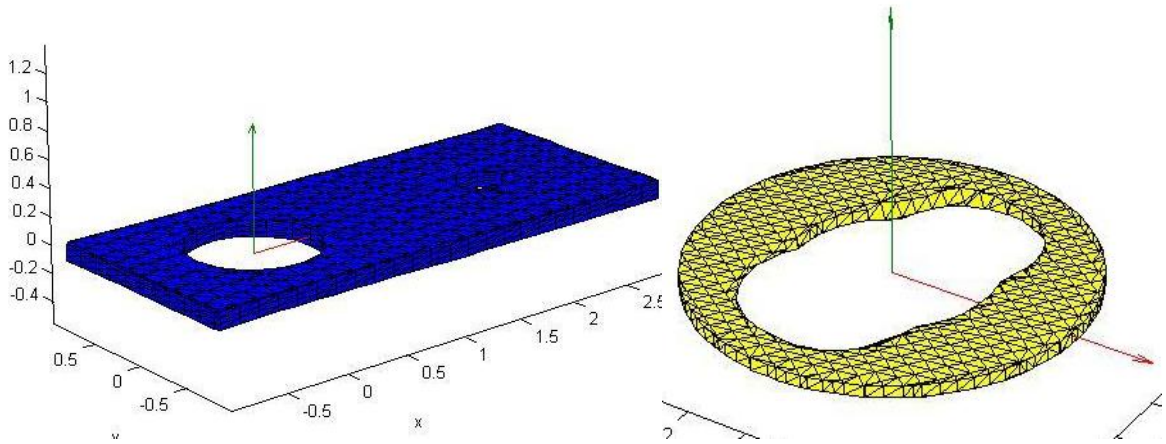


Figure 14: CSG Objects with Two Orthogonal Vectors
A [1, 0, 0] (Red) and Vector *B [0, 0, 1]* (Green)

While the static CSG object was created, an addition property was added. It is comprised of two points, P_{L1} and P_{L2}, located through the object. These two points are appended to the static object's vertices. This will later be used to form an alignment line for the active CSG object as seen in figure 15.

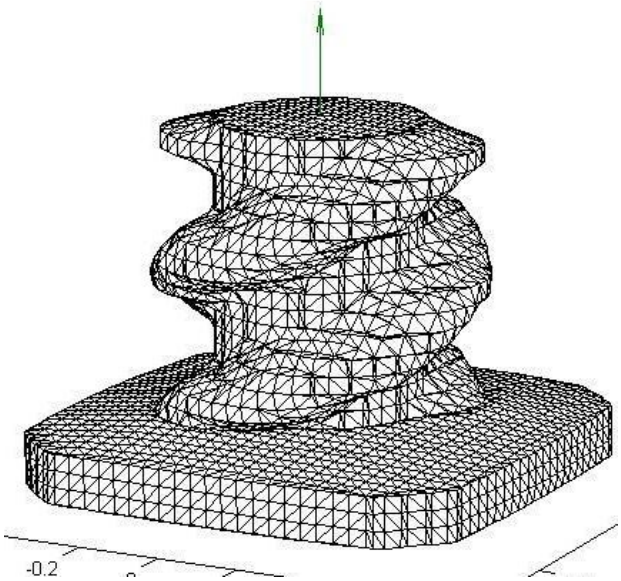


Figure 15: Screw CSG Object with Alignment Line
From Points P_{L1} *[0, 0,-.8]* to P_{L2} *[0, 0, 1] (Green)*

Translation and Rotation of 3D Objects with MATLAB

Active CSG objects can transform via translation or rotation. Any manipulation done to the active CSG object will modify all its vertices points. Thus to translate the object, all the vertices points are summed with the same vector T[x_a, y_a, z_a]. This can easily be done by a simple MATLAB vector addition as shown under equation (1).

$$V_{out}[x', y', z'] = V_{in}[x, y, z] + T[x_A, y_A, z_A] \qquad (1)$$

In this thesis, two methods of rotation are used. The first method incorporates Euler angle rotation. This method of rotation is used to initially randomize the CSG objects in 3D space for testing purposes. It requires all the vertices points to be multiplied by a matrix vector *R* that varies depending on the type of rotation from a given axis. See equations (2)(3) and (4) for various rotation matrix *R* values where V_{in} represents the vertice points of the pre-rotated CSG object, V_{out} represents the vertice points of the post-rotated CSG object, and Θ represents the angle to rotate by in radians.

To rotate along the X axis:

$$V_{out} = R_{x\text{-}axis} \; x \; V_{in} \quad \text{where} \quad R_{x\text{-}axis} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad \text{for some angle } \theta \qquad (2)$$

To rotate along the Y axis:

$$V_{out} = R_{y\text{-}axis} \; x \; V_{in} \quad \text{where} \quad R_{y\text{-}axis} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad \text{for some angle } \theta \qquad (3)$$

To rotate along the Z axis:

$$V_{out} = R_{z\text{-}axis} \; x \; V_{in} \quad \text{where} \quad R_{z\text{-}axis} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{for some angle } \theta \qquad (4)$$

After the active CSG objects have been created with the translation and Euler angle rotation properties, the objects are randomly translated away from the origin point in a spherical pattern (some radius distance away) and rotated in all axis rotations by random angles. This transformation will cause the unit vectors *A [1, 0, 0]* and *B [0, 0, 1]* to now be represented as *A'* and *B'*. This will allow the AI system to be tested to see how well it can maneuver the active CSG object to its goal transformation and rotational orientation.

For testing, the active CSG objects will be given the ability to rotate via axis angle rotation. This will allow the objects to avoid the Gimbal Locking problem and give the objects the ability to rotate in a screw like fashion. There are four parts to this method of rotation. The active CSG object's CGRP must first be translated to the 3D space's origin. This will allow the object to properly rotate around itself instead of rotating around the 3D space's axes and convert *A'* and *B'* back into unit vectors

To handle multiple axis rotations, 3D objects are represented with two vectors. This is because one vector can only represent two of the three rotation orientation of an object. The second vector will provided the additional rotation orientation of an object. The next two parts of the axis angle rotation will then use the two vectors *A'* and *B'* to properly rotate the object.

In parts two and three, the current unit vectors *A'* and *B'* will be rotated to the desired unit vectors *A [1, 0, 0]* and *B [0, 0, 1]* sequentially. This is illustrated under figure 16. Vectors *A'* and *B'* values get compared with the desired vectors *A* and *B* respectfully via MATLAB's vrrotvec function. This function calculates the axis angle vector V_A , vector V_B , angle to rotate Θ_A, and angle to rotate Θ_B. Each axis of rotation vector is determined by the cross product of the two vectors while the angle is determined by the dot product of the two vectors as shown in equations (5) and (6).

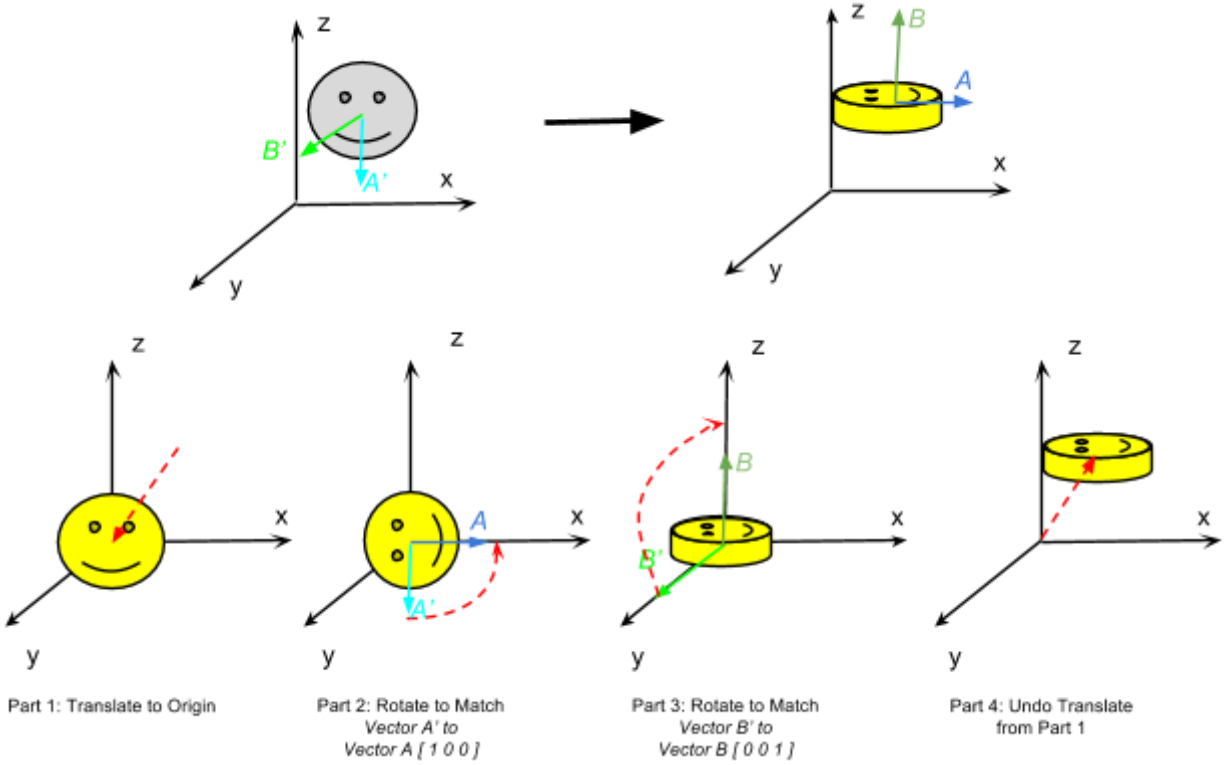


Figure 16: Steps For Axis Angle Rotations

$\Theta_B = \text{acos}(A' \bullet A)$ $\Theta_B = \text{acos}(B' \bullet B)$ or rand()*pi-pi/2; (5)

$V_A = \text{norm}(A' \times A)$ $V_B = \text{norm}(B' \times B)$ (6)

Note the following:

> "If the vectors are parallel (angle = 0 or 180 degrees) then the length of *A* x *A'* will be zero because sin(0)=sin(180)=0. In the zero case the axis does not matter and can be anything because there is no rotation round it. In the 180 degree case the axis can be anything at 90 degrees to the vectors so there is a whole range of possible axes."[Baker 2]

To rotate the active CSG object, all vertices are translated from one point to another at different rates. This is done with the help of Olinde Rodrigues' rotation algorithm shown under equation (7) and (8)

where V_{in} represents the vertice points of the pre-rotated CSG object, V_{out} represents the vertice points of the post-rotated CSG object, V_A and V_B represents the axis angle vector, and Θ_A and Θ_A represents the angle to rotate by in radians.

$$V_{OUT} = V_{IN}\cos(\Theta_A) + V_A(V_A \bullet V_{IN})(1-\cos(\Theta_A) + (V_A \text{ x } V_{IN})\sin(\Theta_A) \quad (7)$$
$$V_{OUT} = V_{IN}\cos(\Theta_B) + V_B(V_B \bullet V_{IN})(1-\cos(\Theta_B)) + (V_B \text{ x } V_{IN})\sin(\Theta_B) \quad (8)$$

The rotation from vector *A'* and *A* from equation (7) will only align the object in two of the three rotation orientation needed. Thus part two gets replicated as part three in equation (8) but with vectors *B'* and *B* (vectors that are orthogonal to vectors *A'* and *A*) to align the last rotation orientation.

Initially, all active CSG objects are rotated by angle Θ_A and vector V_A to partially align the orientation. To further orientate the active CSG object, the object is either directly or randomly rotated by some angle Θ_B and vector V_B. See figure 17 for an illustration of direct or random rotations. Thus equation (5) has two possible values for Θ_B. This is because there are two characterized types of active CSG objects in this thesis. Non-screw-like rotating objects like the rectangles do not need to be constantly rotated to be assembled with the screw. Thus objects like the rectangles are directly rotated to the correct orientation. Screw-like-rotating objects like the nut need to perform a screw like motion to be assembled with the screw. "By definition, it [screw motion] interpolates any two poses by a combination of a minimal angle rotation with a shortest vector translation. Note that the rotation axis is parallel to the translation vector" [Kim 11]. See figure 18 for an illustration of a screw motion. Thus objects like the nut are rotated randomly by random angles Θ_B to allow the objects to randomly find its orientation needed to move itself to the goal location.

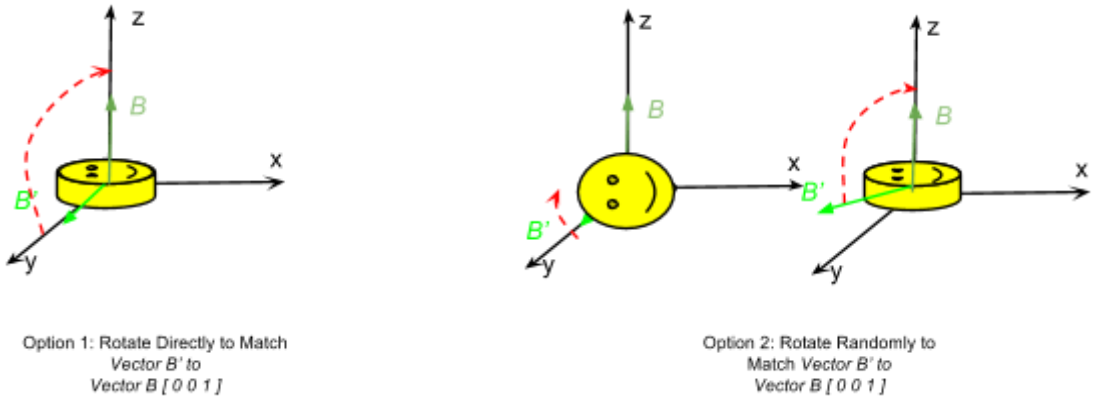


Figure 17: Axis Angle Rotations Can Either Rotate Directly or Randomly.

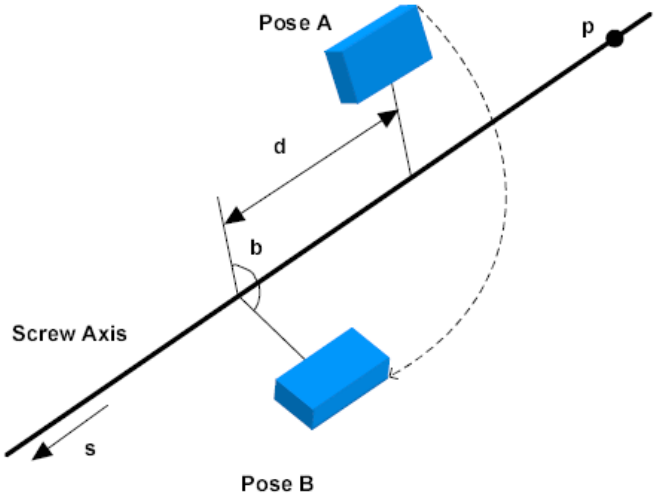


Figure 18: Illustration of a Screw Motion Comprised of Both Translation and Rotation[7]

Part four is the reversal of part one, to translate the newly rotated object back to its original position.

[7] Kim, Byungmoon, and Jarek Rossignac. "Collision prediction for polyhedra under screw motions." *Proceedings of the eighth ACM symposium on Solid modeling and applications* 16 Jun. 2003: 4-10.

Collision Detection

When working with translating and rotating objects in the same 3D space, objects must be restricted from sharing the same coordinate points as each other. Thus a collision detection algorithm was implemented. However, performing a collision detection check requires many CPU cycles when dealing with 3D objects. The worst computational scenario is when two objects have not collided. To verify no collisions have occurred, all the vertices combinations of the two objects must be evaluated. The collision detection algorithm is broken into two evaluation case of non-collision and collision scenarios.

To further understand how the collision detection algorithm is designed, a scenario in which a ball moves through a box has been created in MATLAB with an Intel 2.4Ghz Duo Core Computer. See figure 19 for a visual of this scenario. Under figure 20, the results of the evaluation time have been plotted for a normal collision detection algorithm. It took about 0.3439 seconds of computation time for no collisions to have occurred and about 0.01085 seconds of computation time when a collision did occur.

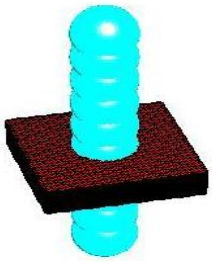


Figure 19: Normal Collision Detection Test

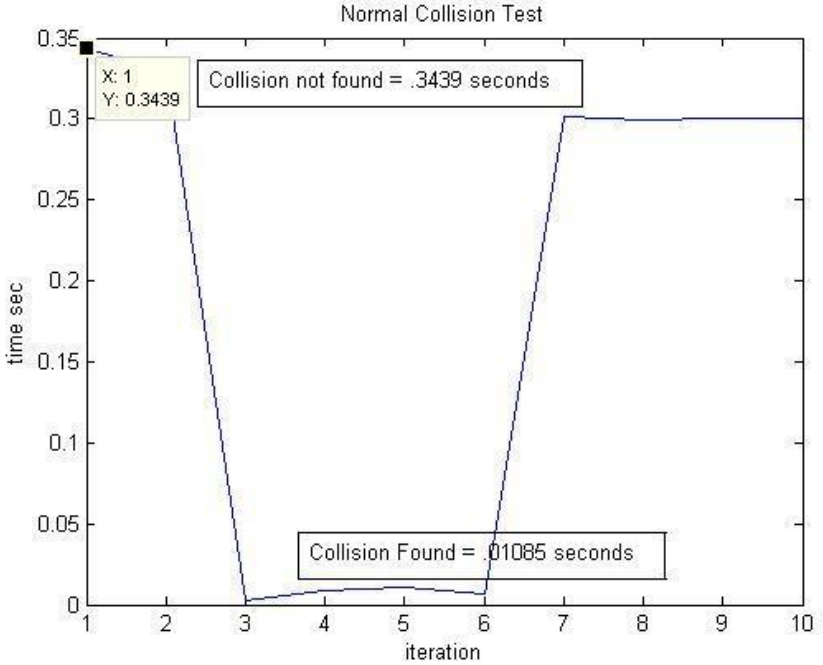


Figure 20: Evaluation Times for the Normal Collision Detection

A preliminary test was added to the Normal Collision Detection Algorithm to lower the non-collision detection time. Each object is generalized with a rectangular bound.

Given objects A and B with the following variables representing their minimum and maximum values:

ax_min, ay_min, az_min, ax_max, ay_max, and az_max
bx_min, by_min, bx_min, bx_max, by_max, and bz_max

The objects may (this must be further evaluated) have collided if the equations (9), (10), and (11) are true:

ax_min < bx_max and ax_max > bx_min (9)
ay_min < by_max and ay_max > by_min (10)
az_min < bz_max and az_max > bz_min (11)

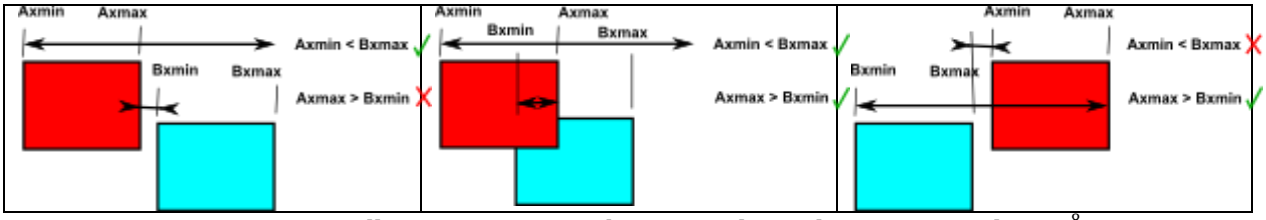


Figure 21: Collision Detection with Rectangular Preliminary Bounds Test[8]

In the case in which the bounds do not overlap, the collision detection can safely assume the objects have not collided. See figure 21 for an illustration of the preliminary bound test. If the bounds do overlap, the next case is used to evaluate collisions.

In the case in which the bounds do overlap, objects' vertices points are evaluated to check if they are too close to each other instead of being equal to each other. Because the objects are represented by numerous vertices points, it may be possible to have CSG objects overlap (a collision has occurred) but not their vertices points. Thus each vertex point for each CSG object needs to be a small distance apart from each other. The distance is evaluated with the Euclidean Distance formula as shown in equation (12). If any vertex point from object A to object B has a distance smaller than some user defined threshold, a collision is determined to have occurred.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}\ . \qquad (12)$$

With the rectangular preliminary collision detection algorithm implemented, figure 22 illustrates the evaluation time to have an advantage over the normal collision detection algorithm.

[8] Baker, Martin John. "EuclideanSpace." 2003. 10 Nov. 2014
<http://www.euclideanspace.com/maths/algebra/vectors/angleBetween/>

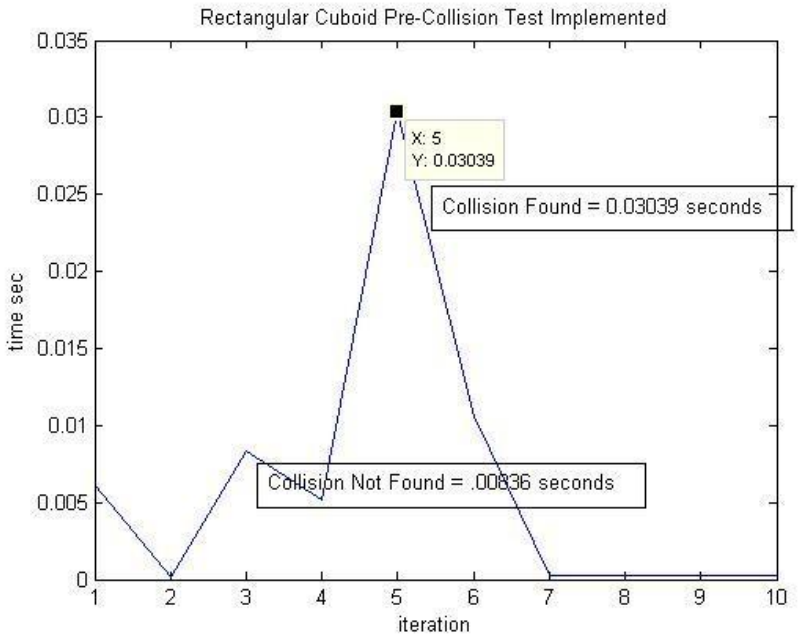


Figure 22: Rectangular Preliminary Collision Detection with Euclidean Distance

With the rectangular preliminary collision detection algorithm, it took about 0.03039 seconds of computation time for the non-collision scenario and about 0.00836 seconds of computation time for the collision scenario. This led to an algorithm with a 410% faster time for the non-collision scenario but 30% slower time for the collision scenario when compared to the normal collision test.

To further improve the algorithm's collision scenario, MATLAB's random permutation function is used. A normal collision detection algorithm will use an evaluation for loop to compare a given vertex point from one CSG object to a matrix of vertices points of the other CSG object. This evaluation for loop will index the vertices points of the first CSG object in order. However, implementing a random permutation indexing for loop instead of a normal ordered indexing for loop will provide a more spread out generalization of first CSG object sooner. This occasionally may provide a faster collision scenario time.

It is important to note, the collision detection algorithm will ignore the additional properties added to the active and static CSG objects. The vertices from the object's CGRP, *vector A*, *vector B*, alignment points P_{L1} and P_{L2} were not evaluated by the collision detection algorithm.

Genetic Algorithms

It is necessary for the AI system to be able to dynamically relearn how to transform a given active CSG objects after a previous transformation has occurred. Implementing a GA will provide this dynamic property. The GA will provide the AI system with the ability to select ideal transformation options from a mating pool as well as explore more options via mutations.

The AI system's mating pool is comprised of twenty transformation options supplied by the weight schema for each generation of the GA. Each transformation consists of translation values in the X Y Z direction, axis angle rotation values, as well as the sum of the weights as the transformation score.

To provide more exploratory transformation options, the GA can also provide a mutation property to relocate the active CSG object to some radius distance away, in a spherical pattern, from the GLP. If the active CSG object's current route to the goal location is not favored by the system, the AI will eventually use mutation to relocate the active CSG object to find a new route to the GLP. Each transformation option from each generation has a three percent chance of performing this mutation. Thus the mating pool can include both new transformation options from the weight schema and mutated transformation options.

During each generation, the mating pool is sorted in an ascending manner of transformation scores. The GA then randomly selects a transformation option based off a standard normal distribution with a mean of 1, sigma of 6, and a range from one to twenty (the mating pool size) as shown under figure 23 and equation (13).

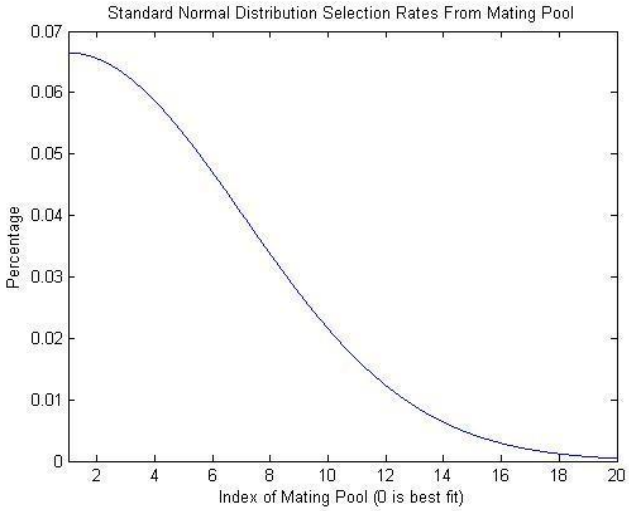


Figure 23: Selecting A Transformation Option Via Standard Normal Distribution

```
pool_size = 20;
while select<1 || select>pool_size
        select = floor( 3.*randn()+1 );
end                                                    (13)
```

If the selected transformation option gets approved from the collision detection algorithm, the active CSG object is transformed. If the active CSG object has not yet transformed to the goal location or a collision has occurred, the GA will continue to the next generation.

For the next generation, the mating pool is emptied except for the selected option and the two lowest scoring transformation options. By passing some transformation option to the next generation, a natural selection concept is kept within the AI system. The weight schema will repopulate the mating pool and the cycle will continue. See figure 24 for an illustration of the GA system.

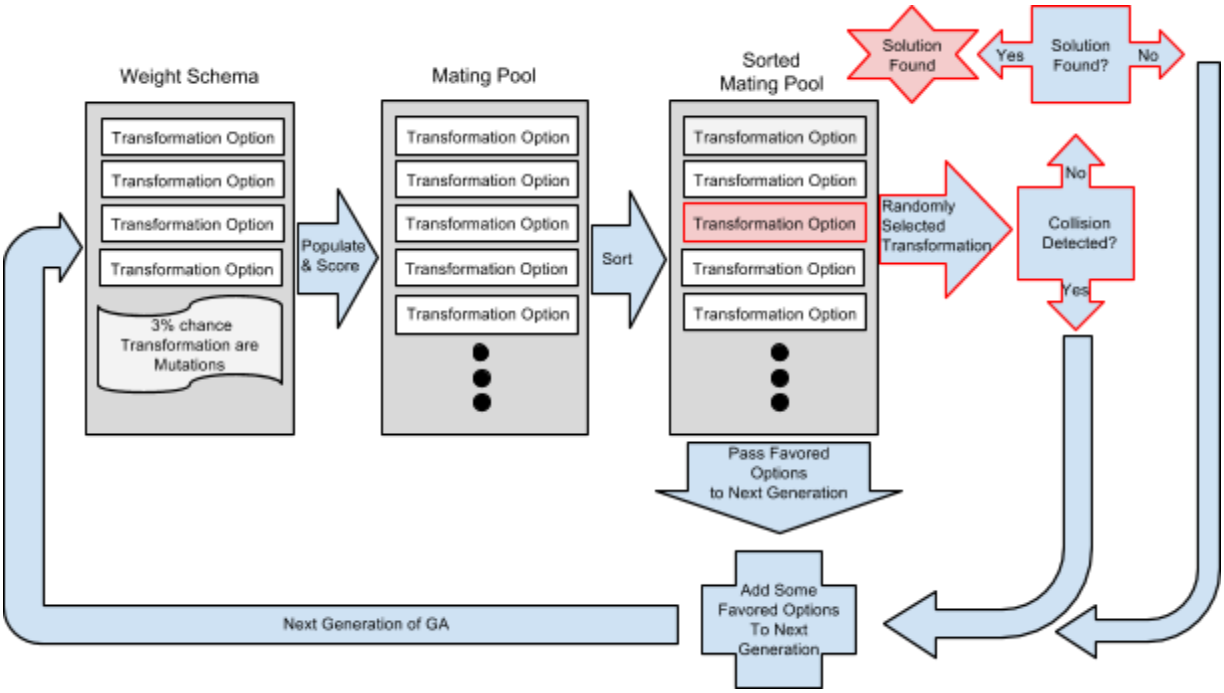


Figure 24: Illustration of the Genetic Algorithm

Weight Schema

In this AI system, active CSG objects have the ability to translate in the X, Y, Z axis and rotate by an axis angle rotation. To allow the system to dynamically learn, a weight schema has been implemented to persuade the active CSG objects to transform themselves towards their GLP. The weight schema is comprised of three parts that can be described as the translation weight, the alignment weight, and the rotational option. The sum of the weights then forms the transformation score. The lower the transformation score, the more accurate the transformation becomes. The rotational option is not part of the transformation score. This is because screw-like rotations need to be randomized and not have a natural selection concept to the weight.

The first part of the weight schema is based on the difference between the CGRP and GLP multiplied by some random positive number, ranging from zero to one. As the distance of the two points becomes closer together, the lower the transformation score becomes. See equation (14) for the definition of each axis' translation weight. But having an overly large translation weight is undesired. Therefore, all the weights have a maximum value as shown under equation (15). In this thesis, the maximum value is set to 0.3. This will limit the active objects from jumping into the desired location without properly moving there. Figure 25 is a two dimensional (2D) representation of the translation weight schema.

```
Next_Generation_X_Translation_Weight = rand() * [CGRP(1) - GLP(1)];
Next_Generation_Y_Translation_Weight = rand() * [CGRP(2) - GLP(2)];
Next_Generation_Z_Translation_Weight = rand() * [CGRP(3) - GLP(3)];          (14)
```

```
maximum_value = 0.3;
if (Next_Generation_X_Translation_Weight > maximum_value);
    Next_Generation_X_Translation_Weight = maximum_value;
if (Next_Generation_Y_Translation_Weight > maximum_value);
    Next_Generation_Y_Translation_Weight = maximum_value;
if (Next_Generation_Z_Translation_Weight  > maximum_value);
    Next_Generation_Z_Translation_Weight = maximum_value;                    (15)
```

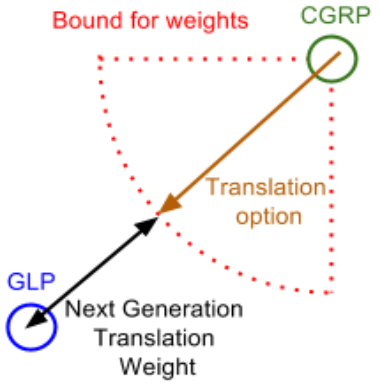


Figure 25: Illustration of Translation Weights Being Bounded

The multiplied random positive number will provide several possible transformation options to be generated but ensure the active CSG object will move towards the GLP. This can be seen under figure 26.

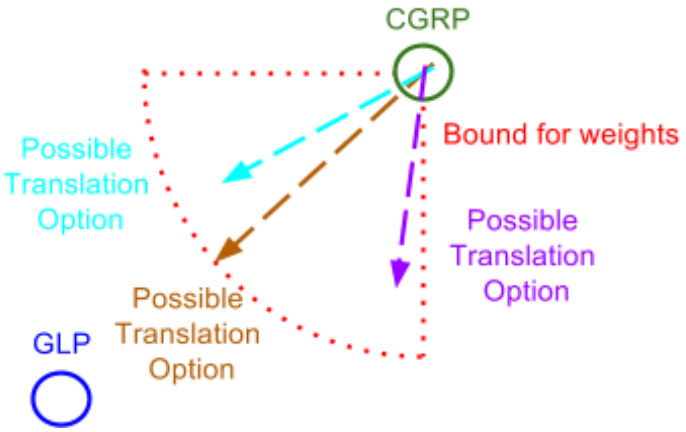


Figure 26: Illustrations of Multiple Translation Options

To further assist the AI system, the concept of aligning the active CSG objects was implemented as the second part of the weight schema. This will allow the active CSG object to position itself before moving or rotating itself onto the static CSG object. To align the active CSG object to the static CSG object, an alignment line was created. This can be constructed from the two appended points, P_{L1} and P_{L2}, that were created on the static CSG object. The active CSG's CGRP is reused as an evaluation point for evaluating how far the active CSG object is from the alignment line. The distance can then be calculated using the distance from a point to line formula (16) and be used as the alignment weight as shown under figure 27 and equation (16).

$$d = \frac{norm\Big(cross\big((X_0 - X_1),(X_0 - X_2)\big)\Big)}{norm(X_2 - X_1)} \quad (16)$$

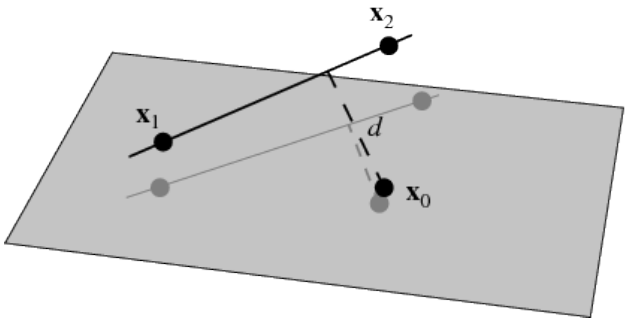


Figure 27: Image of Distance (d) From Point (X_0) to a Line (X_1 X_2)[9]

$$\text{Alignment Weight} = \frac{norm\Big(cross\big((CGRP - P_{L1}),(CGRP - P_{L2})\big)\Big)}{norm(P_{L1} - P_{L2})} \quad (17)$$

The sum of the two weights (translation weight and alignment weight) forms the transformation score. The alignment weight can then be biased to be higher priority by multiplying it by some value higher than

[9] "Point-Line Distance--3-Dimensional - Wolfram MathWorld." 2002. 15 Nov. 2014 <http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html>

one. In this AI system, the alignment weight has five times the influence towards a score than the translation weight as shown under equation (18) and figure 28.

$$\text{Transformation Score} = \text{Translation Weight} + (5 * \text{Alignment Weight}) \quad (18)$$

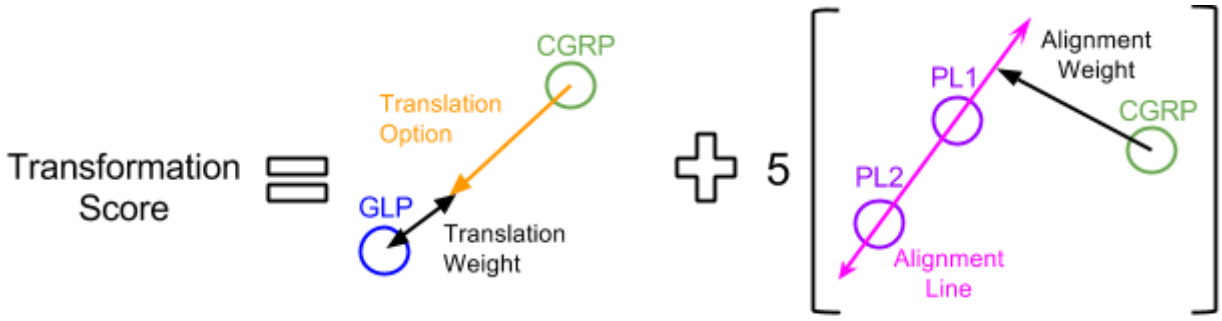


Figure 28: Illustration of Equation for Transformation Score

The third part of the weight schema is based on a rotation option to rotate the active CSG object. All active objects are initially rotated by some angle Θ_B via axis angle vector V_A. Depending on the type of active CSG object being evaluated, the following transformations can either orientate the object directly or randomly to the desired location via angle Θ_B and vector V_B. The transformation score is not affected by the rotation option.

These three parts to the weight schema will produce transformation options as shown in figure 28 for the GA mating pool.

X Translation	*Y Translation*	*Z Translation*	*X axis of* V_B	Y axis of V_B	Z axis of V_B	Θ_B *(radians)*	*Transformation Score*

Figure 28: A Transformation Option From The GA Mating Pool

Video Results

I. A ball finding its way into a cup
   A. Video of the AI system manipulating the objects
      1. https://drive.google.com/file/d/0B5N7gcjhbnlmMTRjLXFCZTkyN00/view?usp=sharing

II. Active rectangles with a static screw
   A. Creation of the objects
      1. https://drive.google.com/file/d/0B5N7gcjhbnlmMWw2YTFjaHZZdkk/view?usp=sharing
      2. https://drive.google.com/file/d/0B5N7gcjhbnlmQy1fVW5sS29MVlk/view?usp=sharing
      3. https://drive.google.com/file/d/0B5N7gcjhbnlmR1lWM3lMbkRjTFE/view?usp=sharing
   B. AI system manipulating the objects
      1. https://drive.google.com/file/d/0B5N7gcjhbnlma3VLTFl3b1lsTjA/view?usp=sharing
      2. https://drive.google.com/file/d/0B5N7gcjhbnlmZGw2d1psX0JxdTA/view?usp=sharing
      3. https://drive.google.com/file/d/0B5N7gcjhbnlmcTdFbDJySnVncUE/view?usp=sharing

III. Active nut with a static screw
   A. Creation of the objects
      1. https://drive.google.com/file/d/0B5N7gcjhbnlmaWNDc0ZLZm91dTA/view?usp=sharing
   B. AI system manipulating the objects
      1. https://drive.google.com/file/d/0B5N7gcjhbnlmaldMLXVUa3B4Y2c/view?usp=sharing

IV. Active rectangles and active nut with a static screw
   A. Creation of the objects
      1. https://drive.google.com/file/d/0B5N7gcjhbnlmUTh6bFZCWnJNckU/view?usp=sharing
   B. AI system manipulating the objects
      1. https://drive.google.com/file/d/0B5N7gcjhbnlmN1VjaXZmZlJRVUU/view?usp=sharing
      2. https://drive.google.com/file/d/0B5N7gcjhbnlmZ2F2U1ZzRXUzUlk/view?usp=sharing
      3. https://drive.google.com/file/d/0B5N7gcjhbnlmQ2lzd2R3ZGIwUlk/view?usp=sharing

Conclusion

In this thesis, a weighted GA AI system was implemented to provide an AI with spatial intelligence. Given a few 3D objects, the AI was able to maneuver the objects so that they may be assembled with each other. This methodology may allow future AIs to learn to interact with the 3D world for themselves as we have done so as toddlers. This may also allow the AI to learn to improve its own common sense ability and score higher on the standard IQ test. Robert Sloan is quoted as saying, "We're still very far from programs with common sense and artificial intelligence that can answer comprehension questions with the skill of a child of eight." [Arnold 1]

References

1. Arnold, Stephen E."Where Is ConceptNet, Watson? - KMWorld Magazine." 2013. 5 Nov. 2014 <http://www.kmworld.com/Articles/News/News-Analysis/Where-Is-ConceptNet-Watson-92666.aspx>
2. Baker, Martin John. "EuclideanSpace." 2003. 10 Nov. 2014 <http://www.euclideanspace.com/maths/algebra/vectors/angleBetween/>
3. Dinis, Rafael, Anabela Simões, and Jorge Bernardino. "GraphEA: a 3D educational tool for genetic algorithms." *Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*. ACM, 2013
4. Diochnos, Dimitrios I. "Commonsense Reasoning and Large Network Analysis: A Computational Study of ConceptNet 4." *arXiv preprint arXiv:1304.5863* (2013).
5. Faure, François et al. "Image-based collision detection and response between arbitrary volume objects." *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* 7 Jul. 2008: 155-162.
6. Hadap, Sunil et al. "Collision detection and proximity queries." *ACM SIGGRAPH 2004 Course Notes* 8 Aug. 2004: 15.
7. Land, Bruce R. "Hierarchical graphics modeler using MATLAB." 20 Oct. 2014 <http://www.nbb.cornell.edu/neurobio/land/PROJECTS/Hierarchy/>
8. Gaudin, Sharon. "Top Artificial Intelligence system is as smart as a 4-year-old " 2013. 18 Oct. 2014 <http://www.computerworld.co.nz/article/520936/top_artificial_intelligence_system_smart_4-year-old/>
9. Ghose, Tia. "How Smart Is Advanced Artificial Intelligence? Try Preschool Level" 2013. 18 Oct. 2014 <http://www.livescience.com/38310-ai-has-four-year-old-iq.html>
10. Holland, John H. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.*. U Michigan Press, 1975.
11. Kim, Byungmoon, and Jarek Rossignac. "Collision prediction for polyhedra under screw motions." *Proceedings of the eighth ACM symposium on Solid modeling and applications* 16 Jun. 2003: 4-10.
12. Morgan, Matthew JW, and Christine L Mumford. "A weight-coded genetic algorithm for the capacitated arc routing problem." *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* 8 Jul. 2009: 325-332.
13. Ochiai, Hiroyuki, and Ken Anjyo. "Mathematical basics of motion and deformation in computer graphics." *ACM SIGGRAPH 2014 Courses* 27 Jul. 2014: 19.

14. Rossignac, Jaroslaw R. "Constraints in constructive solid geometry." *Proceedings of the 1986 workshop on Interactive 3D graphics* 1 Jan. 1987: 93-110.