The lua-tikz3dtools Package

The lua-tikz3dtools Package, Version 1.1.0

https://github.com/Pseudonym321/TikZ-Animations/tree/master1/TikZ/lua-tikz3dtools

Jasper Nice

September 4, 2025

Dedicated to those who, like me, never found the tools they needed—and chose to build them.

“I long to accomplish great and noble tasks but it is my duty to accomplish small tasks as though they were great and noble.” —Sylvia Fedoruk

Contents

Acknowledgements **ix**

1 Problem Statement **1**
1.1 The Core Problem being Addressed 1
1.2 Importance of the Issue 1
1.3 Intended Audience 2

2 Literature Review **3**
2.1 Existing Approaches in Practice and Academia 3
2.2 Limitations and Open Challenges in Current Methods 3
2.3 Proposed Approach and Its Advantages Over Existing Work 4

3 Methodology **5**
3.1 Detailed Description of the Proposed Approach 5
3.1.1 Inverse and Basis-Relative Affine Transformations: The Camera 5
3.1.2 The Comparator for Geometric Primitive Occlusion 5
Point Versus Point 6
Point Versus Line Segment 6
Point Versus Triangle 7
Line Segment Versus Line Segment 7
Line Segment Versus Triangle 7
Triangle Versus Triangle 8
3.1.3 The User Interface 8
Review of Projective Transformations 8
The Projective Matrix Library 8
The Zero-Dimensional Point 9
The One-Dimensional Curve 9
The Two-Dimensional Surface 10
The Three-Dimensional Solid 10
Ordering and Displaying the Generated Primitives 11
3.2 Rationale and Development Process of the Approach 11
3.3 Challenges Encountered and Solutions Implemented 12
3.4 Remaining Challenges and Directions for Future Work 12

List of Figures

1.2.1 Inconclusive results are necessary for the transitivity of the comparator. We can only definitively compare primitives which are in a direct occlusive relationship. Even points must have their occlusive relationship neglected if they do not directly overlap on the viewing plane. In this figure, A occludes B and B occludes C, but A and C have no occlusive relationship, even though C is higher than A. If we didn't make this case inconclusive, then the transitivity would be immediately broken. 2

Acknowledgements

This work was typeset using TEX, the typesetting system created by Donald E. Knuth, along with various extensions and packages developed by the TEX community. I am grateful to the vibrant TEX Stack Exchange community for their ongoing support and resources. For those interested, my contributions can be found at Jasper [Jasa].

Jasper Nice

Chapter 1

Problem Statement

1.1 The Core Problem being Addressed

The package `lua-tikz3dtools` aims to provide 3D illustration capabilities to the LaTeX ecosystem. In particular, there is a focus on both camera movement and perspective, as well as on rigorous occlusion. There are already LaTeX packages which provide 3D illustration capabilities. To the author's knowledge, there is no current algorithm for taking a set of parametrically defined primitive, clipping them in such a way that cyclic overlaps and intersections are totally eliminated, and occlusion-sorting the resulting non-intersecting and non-cyclically overlapping primitives. A geometric primitive is a point, a line segment, a triangle, a tetrahedron, or any higher dimensional simplex. The primary objective of `lua-tikz3dtools` is to implement such a routine, as well as to use known methods to enable the feature of arbitrary camera positioning. Currently, both arbitrary camera positioning and occlusion of non-intersecting and non-cyclically overlapping primitives up to triangles are implemented. Automatic clipping of primitives which do not fall into this category is still aa planned feature. To the author's knowledge, no such algorithm exists yet.

1.2 Importance of the Issue

Most 3D illustration softwares use a process called z-buffering, in which a pixel-level resolution is achieved by tracing numerous orthogonal lines off the viewing plane until they reach the first geometric primitive along their trajectory. It really only works for triangles, because line segments and points are overly determined and will rarely intersect the orthogonal lines. As such, the visualization of parametric curves and points is neglected. The method described in this book for the occlusion of non-intersecting and non-cyclically overlapping geometric primitives is a transitive comparator which does not always produce true or false. As a matter of fact, occlusion tests are often inconclusive—and they need to be for transitivity to exist! For example, consider the extremely simple set of primitives, ordered along the y-axis; the point $A(0, 0.25)$, the line segment $B\{(0, 0), (1, 1)\}$, and the point $C(1, 0.75)$ (Figure 1.2.1). The

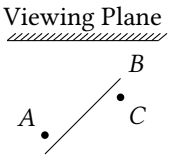

Figure 1.2.1: Inconclusive results are necessary for the transitivity of the comparator. We can only definitively compare primitives which are in a direct occlusive relationship. Even points must have their occlusive relationship neglected if they do not directly overlap on the viewing plane. In this figure, A occludes B and B occludes C, but A and C have no occlusive relationship, even though C is higher than A. If we didn't make this case inconclusive, then the transitivity would be immediately broken.

author has the sincere impression that give a set of non-intersecting and non-cyclically overlapping primitives, there should be a definitive method of deciphering occlusive order among those primitives. The author also sincerely believes that primitives which do intersect and do cyclically overlap can be partitioned such that they no longer do so. The author sincerely believes that such a method should be considered as a fundamental concept in 3D graphics, and is sincerely surprised that such a transitive comparator for 1–3 dimensional affine simplexes has never been the subject of serious formal study. Compared to z-buffering, the proposed algorithm is incredibly slow, but this can be mitigated by culling primitives which are totally occluded by other primitives. In plain language, the proposed comparator is an enormous leap forward for occlusion sorting tessellated parametric objects. Additionally, while camera positioning is well conceived in computer graphics already, it has never yet been implemented in an illustration tool from the TEX ecosystem. The package is still under development, and will be no longer experimental when the automatic clipping is both implemented and field tested.

1.3 Intended Audience

This work is intended for mathematics illustrators who want to make 3D illustrating using a LATEX-cohesive package. `lua-tikz3dtools` is capable of rendering perspective scenes with arbitrary camera positioning and rigorous occlusion sorting. It is presumed that users will already be aware of how to define parametric objects, as well as with projective transformation matrices. The author suggests the first edition of the book by Rogers and Adams for an exemplary introduction to projective matrices.

Chapter 2

Literature Review

2.1 Existing Approaches in Practice and Academia

z-buffering is a contemporary tool for approximating occlusion of triangles by taking a pixel-level raster of rays from the viewing plane onto what lies before it. The colour of the first triangle in the path of such a ray is assigned to the pixel of the ray. There is also the painter's algorithm in which primitives are layered from back to front, though it is a very general description [Nyb11]. One variant of the painter's algorithm in the context of parametric renderings is the centroid sort, in which geometric primitives are ordered by the depth of the average of their vertices [Ski25]. There are two ways to find these depths; one way is to take the dot product with the observer vector [httb]. Another way is to use an affine transformation to align the z axis with the observer vector, and then order by just that one component.

Setting the camera position aand orientation is done by determining the affine transformation matrix which carries the camera to the desired position, and performing the inverse of that transformation on all of the parametric objects. Perspective is achieved by using a projective transformation after all the affine transformations are done. While this practise is commonplace in 3D graphics, it has yet to be achieved using tools in the TEX ecosystem.

2.2 Limitations and Open Challenges in Current Methods

z-buffering is fundamentally an approximation. While it is useful for real-time 3D graphics, the author of this book contends that aa definitive occlusion sorting of non-intersecting and non-cyclically overlapping primitives should be considered as a fundamental concept in 3D occlusion. The centroid sort on the other hand is prone to non-subtle occlusion errors [Jasb]. In attempt to resolve this, I tried separating the triangles into groups based on the orientation of their normal vector with respect to the observer's direction, and then performed a centroid sort on both groups. This

method was still insufficient.

2.3 Proposed Approach and Its Advantages Over Existing Work

This package proposes a transitive occlusion comparator for non-intersecting and non-cyclically overlapping geometric primitives. This comparator is designed to be used alongside an automatic primitive-primitive clipping algorithm, and with a more sophisticated variant of back-face culling which accounts for translucency. The future goal of this package is to introduce the aforementioned geometric primitive clipping software to complement this sorting algorithm. Possibly, another goal is to implement such a back-face culling algorithm as well, though that is secondary to the current goal. The advantage of the occlusion comparator introduced in this package is its clarity. Unlike z-buffering, this method offers a definitive ordering of non-intersecting and non-cyclically overlapping geometric primitives. It is paired with an artificial inteligence-written topological sort function which is capable of ordering the primitives based on the comparator's output.

Chapter 3

Methodology

3.1 Detailed Description of the Proposed Approach

3.1.1 Inverse and Basis-Relative Affine Transformations: The Camera

Camera positioning is to be achieved through inverse affine transformations. In particular, we take the desired camera transformation, and we transform the world by its inverse, leaving the viewing rectangle stationary. The viewing rectangle is in a very literal sense the computer screen. When you're scrolling through an article on your screen, the screen itself is not moving up and down, but the article is moving relative to the screen. Transformations within the camera's viewing rectangle are performed using basis-relative affine transformations. These transformations make use of matrix similarity to represent transformations in one affine basis in terms of another affine basis. An affine basis is a set of basis vectors, along with a point of origination; they are a means of navigating and traversing vector space. An affine basis in $\mathbb{R}^3$ may be zero dimensional, all the way up to three dimensional. It's unit-interval vectors are linearly independent, and they form a simplex with the point of origination. A primary feature of the camera view is the culling of primitives which fall behind it. When a person looks forward, they do not see objects behind them—`lua-tikz3dtools` endeavors to emulate this feature. For reasons of speed, primitive culling will eventually be implemented by setting the camera frame explicitly before rendering, so that not only just the primitives behind the camera don't show up, but neither do the ones which fall beyond its scope in general.

3.1.2 The Comparator for Geometric Primitive Occlusion

`lua-tikz3dtools` introduces a novel comparator for geometric primitive occlusion. Given a set of non-intersecting and non-cyclically overlapping primitives, this algorithm provides a partial transitive ordering of the primitives. This is possible due to the deliberate implementation of inconclusive results. That is, the comparator only returns aan ordering for two given primitives if and only if their orthogonal

projections on the viewing plane overlap—meaning that there is an orthogonal ray off the plane which intersects both. They are ordered by the points of intersection of this exact ray and both primitives. There are three primitives, and each one can be compared to either one of its own, or another. This presents up with six unique possible comparisons:

1. point versus point,
2. point versus line segment,
3. point versus triangle,
4. line segment versus line segment,
5. line segment versus triangle, and
6. triangle versus triangle.

In Figure 1.2.1, the comparison between A and C is inconclusive, because despite the fact that C is nearer to the viewing plane than A, they are not in a direct occlusive relationship. As mentioned in the problem statement, introducing definitive results for primitives which are not in a direct occlusive relationship would obliterate the comparator's transitivity, which is absolutely essential for the function which sorts primitives based on the comparator's results.

Point Versus Point

For the reason of numerical instability, we have a difference threshold of 0.001 centimeters, within which two points are considered identical. This was not an arbitrarily chosen number; it was chosen because illustrations for human eyes do not need to be more precise than one one thousandth of a centimeter. Hence, two points within this distance of each other are compared to produce an inconclusive result. Next, should they pass the first test, the points are projected onto the viewing plane orthographically. The distance of their projections is subjected to the same litmus test, only this time they are occluded based on their depth if and only if their projections overlap, because that means that one occludes the other. If they are not in the immediate vicinity of each other, but their projections are, they are occluded based on their depth. An interesting note is that we could have used a much, much, smaller difference threshold; even one one millionth would have been usable, but it would be unnecessary. Maybe it is possible that I will make it one ten thousandth—but no less!

Point Versus Line Segment

We obtain an affine basis for the line through the line segment by using its start point as the affine origin, and the difference of the second point with the first as the direction vector. We then take the difference of the point and the line's affine origin. Here's the secret of the method: The orthogonal projections of the difference of the point and the line's affine origin, and the affine direction vector, is collinear with the viewing axis if and only if it is not impossible that the point and the line occlude each other!

Of course, we project the point and its projection on the line onto the viewing plane. If these viewing plane projections are within vicinity of one another, then we proceed with the test. There is one final criteria to be tested before we compare depths, and that is whether the points projection falls within the line segment, and not beyond it. We divide their lengths to obtain the orthogonal vector projection's length relative to the affine basis unit interval. If that relative length is between zero and one, then they definitely occlude each other, and we take the inverse projection of that viewing plane point onto both primitives, and we sort by the depths of these inverse projections. If no such occlusive relation is present, then the result is inconclusive.

Point Versus Triangle

The first step of this comparison is to determine if the projection of the point onto the viewing plane falls within the triangle's viewing plane projection. We do this by projecting both primitives onto the viewing plane. We then connect each vertex of the triangles projection by vectors. From the origin of each vector, a second vector is extended to the point. Due to the geometric nature of the cross product, in particular its anticommutativity, if we take the cross products of each pair of vectors originating from the same point, if all of them are parallel but not antiparallel, then the point is inside the triangle. Think about what happens to one of the cross products if the point falls beyond the projected triangle—a cross product will flip. If the result of this experiment is a definitive result that the point's projection does fall within the triangle's projection, then we take the inverse orthogonal (to the viewing plane) projection onto both primitives, then we sort by the depths of these inverse projections. To calculate the vertical projection onto the triangle, we solve for the projected point's coordinates in the projected triangle's affine basis (it is a simplex), and then use those coordinates in the affine basis of the non-projected triangle to obtain the inverse projection.

Line Segment Versus Line Segment

We project the line segments onto the plane, and obtain an affine basis for both. We then take the length of a cross product of the direction vectors. If this length is nonzero, meaning that the lines are noncollinear, then we solve for the coordinate of the intersection in both affine bases. If both coordinates are between zero and one, then we sort by those coordinates on the lines through the original bases. If the cross product is zero, then we perform our comparator for points and line segments on both points of both affine bases on the other affine basis. If all of these are inconclusive, then the result is inconclusive.

Line Segment Versus Triangle

We perform point versus triangle for both points of the line segment with the triangle. We also perform line segment versus line segment between the line segment and each line segment of the triangle. If any of these are conclusive, then we sort based on that, otherwise it is inconclusive.

Triangle Versus Triangle

We check each line segment of one triangle with each line segment of the other. We also check each vertex of both triangles with the other triangle. If any of these are conclusive, then we sort based on that, otherwise the test is inconclusive.

3.1.3 The User Interface

Each user-facing command has a set of parameters to which options can be assigned. Programmers will recognize this as a key-value dictionary. `lua-tikz3dtools` exposes commands for generating and displaying sets of geometric primitives. The system is geared toward illustration of parametric objects, meaning that they are defined using parametric definitions. Currently, `lua-tikz3dtools` supports the generation of parametric objects of zero to three dimensions (i.e., their input is a 0–3 dimensional rectangle-equivalent). Before we analyze each command in-depth, we will first review projective transformations.

Review of Projective Transformations

"The 4×4 homogeneous transformation matrix can be partitioned into four separate sections:

$$\left[\begin{array}{c|c} 3\times 3 & 3\times 1 \\ \hline 1\times 3 & 1\times 1 \end{array}\right].$$

The 3×3 matrix produces a linear transformation in the form of scaling, shearing and rotation. The 1×3 row matrix produces translation, and the 3×1 column matrix produces perspective transformation. The final single element produces overall scaling." [RA76]

`lua-tikz3dtools` exposes the command `\setobject`, which enables the user to create their own lua objects. For example,

```
\setobject[name = {blah},object = {euler(pi/2,pi/3,7*pi/6)}]
```

sets the variable `blah` to be a classic rotation matrix for viewing purposes.

The Projective Matrix Library

`lua-tikz3dtools` is projective-matrix aware. Basic linear algebra—up to matrix multiplications and transformations—is required as a prerequisite from the user in order to follow this portion of the manual. `lua-tikz3dtools` exposes the Lua command `matrix_multiply(A, B)` which multiplies two projective matrices, `A` and `B`. Additionally, the command `transpose(A)` returns the transpose of projective matrix `A`. An inportant operation for camera movement is the matrix inverse, and it is achieved by the command `matrix_inverse(A)` for a matrix `A`. Axis and Euler rotation matrices are implemented, and Rodriguez matrices are planned. The commands are `xrotation(angle)`, `yrotation(angle)`, and `zrotation(angle)`. There is also the command `translate(x,y,z)` which returns a translation matrix for the given displacement. There are also `xscale(scale)`, `yscale(scale)`,

`zscale(scale)`, and `xcale(scale)`. The z and y rotation matrices are composed in the function `euler(alpha,beta,gamma)` which returns a z-y-z Euler angle rotation matrix. `identity_matrix()` is the projective identity matrix in three dimensions.

In addition to providing projective matrix features, there are also some practical parametric functions. There is the command `sphere(longitude,latitude)`, `dot_product(u,v)`, `cross_product(u,v)`, `norm(u)`, `normalize(u)`, and even `stereographic_projection(point)`. An important feature of `lua-tikz3dtools` is that all points are represented in nested-table form in both the Lua implementation, and the user-facing interface. That is, the point $(0, 1, \pi)$ is represented by the Lua matrix `{{0,1,pi,1}}`, where the fourth coordinate is the homogeneous coordinate, and it is always exactly one (1!), except for transiently when perspective projection is applied. This coordinate is solely responsible for translation and perspective transformations [RA76].

The Zero-Dimensional Point

`lua-tikz3dtools` exposes the command `\appendpoint` which accepts a zero-dimensional parametric definition, along with some options. For example;

```
\appendpoint[
    x = {cos(tau/6)}
    ,y = {tau/6}
    ,z = {0}
    ,fill options = {
        fill = red
        ,fill opacity = 0.7
    }
    ,transformation = {euler(pi/2,pi/3,pi/6)}
]
```

appends a singular point to the list of geometric primitives to be later sorted and displayed.

The One-Dimensional Curve

A curve is generated by a one-dimensional parametric definition, along with some options. For example;

```
\appendcurve[
    ustart = {0}
    ,ustop = {1}
    ,usamples = {360}
    ,x = {cos(tau*u)}
    ,y = {sin(tau*u)}
    ,z = {0}
    ,draw options = {
        draw
```

```
        ,red
    }
    ,transformation = {euler(pi/2,pi/3,pi/6)}
    ,arrow tip = {true}
    ,arrow tip options = {fill = green}
    ,arrow tail = {true}
]
```

will generate a circle with an arrow on it.

The Two-Dimensional Surface

A surface is generated using a two-dimensional parametric definition. For example;

```
\appendsurface[
    ustart = {0}
    ,ustop = {1}
    ,usamples = {36}
    ,vstart = {0}
    ,vstop = {1}
    ,vsamples = {18}
    ,x = {sphere(tau*u/36,pi*v/18)[1][1]}
    ,y = {sphere(tau*u/36,pi*v/18)[1][2]}
    ,z = {sphere(tau*u/36,pi*v/18)[1][3]}
    ,fill options = {
        preaction = {
            fill = green
            ,fill opacity = 0.8
        }
        ,postaction = {
            draw
            ,line join = round
            ,line cap = round
        }
    }
    ,transformation = {euler(pi/2,pi/3,pi/6)}
]
```

Will generate a triangulated unit sphere.

The Three-Dimensional Solid

A parametric solid takes three-dimensional input, and makes three dimensional output. For example;

```
\appendsolid[
    ustart = {0}
    ,ustop = {1}
```

```
    ,usamples = {36}
    ,vstart = {0}
    ,vstop = {1}
    ,vsamples = {18}
    ,wstart = {0}
    ,wstop = {1}
    ,wsamples = {9}
    ,x = {(1+w)*sphere(tau*u/36,pi*v/18)[1][1]}
    ,y = {(1+w)*sphere(tau*u/36,pi*v/18)[1][2]}
    ,z = {(1+w)*sphere(tau*u/36,pi*v/18)[1][3]}
    ,fill options = {
        preaction = {
            fill = green
            ,fill opacity = 0.8
        }
        ,postaction = {
            draw
            ,line join = round
            ,line cap = round
        }
    }
    ,transformation = {euler(pi/2,pi/3,pi/6)}
]
```

will generate a solid sphere—mapped from a cube! Of course, the domain may also be not a perfect a square; e.g., `ustop = {tau/6}` would be valid too.

Ordering and Displaying the Generated Primitives

Sorting and displaying is done automatically by the command `\displaysegments`.

3.2 Rationale and Development Process of the Approach

I started by studying the problem of occlusion of non-intersecting and non-cyclically overlapping triangles; this was half a year ago, give or take. I asked about it on the Mathematics Stack Exchange, and received an insightful comment from user lisyarus which said to sort based on the inverse projections of overlapping points on the viewing plane [htta]. The idea was in a non-complete form, and its follow through required substantial innovation on my part, but it was the gemstone which inspired me and made all this possible. Of course, this was back when I was still naive on the subject, so there is an accepted answer which I do not agree with in retrospect—please disregard it as it is not helpful.

Armed with the comment by lisyarus, and a lot of time to think, I eventually went about devising methods of determing points of overlap of projected primitives on the

viewing plane. It actually took me a long time to get here from the comment, but here we are. Interestingly, it took many attempts and I made numerous mistakes before a working algorithm was achieved. However, when I analyzed my mistakes, they led me to devise more conclusive tests. Now the software isn't just for triangles. It is for affine simplexes of dimension zero through three. Maybe one day we will take higher dimensional simplex projections—who knows.

3.3 Challenges Encountered and Solutions Implemented

I faced numerous roadblocks along my path to this achievement. For instance, I had to rewrite the entire software from scratch many times, identifying conceptual mistakes each time, before arriving on a working implementation. For example, using a distance threshold for point versus point to only sort them if their projections on the viewing plane are close took me breaking down the problem and thoroughly analyzing it from every perspective that I could. Additionally, while most of the software is self-written, I took the shortcut of having ChatGPT implement a Gauss-Jordan column-major augmented matrix solver, which would have taken me some time to learn on my own. Eventually I will, as it is a fundamental computational tool, but for the time being I used a crutch. Additionally, the topological sort function was written entirely by AI. What is does it compare each primitive to each other primitive using my comparator, and it orders them by its output. An interesting feature is that it identifies occlusion cycles, orders them all at the saame level relative to everything else, and outputs which exact primitives have cyclic occlusion. Of course, this dream of mine really started after learning how to sort points in 3D, and realizing that sorting by centroids of triangles does not work.

3.4 Remaining Challenges and Directions for Future Work

The main two future goals of `lua-tikz3dtools` are a clipping algorithm to eliminate intersections and cyclic overlaps, and a comprehensive, translucency-aware culling algorithm for primitives which would definitively not show up when the scene is rendered in its totality. Only once these are complete would I consider the `lua-tikz3dtools` package to be mature. Until then the software will remain experimental.

Bibliography

[htta] Jasper (https://math.stackexchange.com/users/1499599/jasper). *Exact Triangle Sorting for Orthographic Rendering of a Triangulated Surface*. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/5063772 (version: 2025-05-17). URL: https://math.stackexchange.com/q/5063772.

[httb] P123 (https://math.stackexchange.com/users/1499599/p123). *How does the dot product give correct depth ordering in orthographic 3D projections?* Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/5062592 (version: 2025-05-06). URL: https://math.stackexchange.com/q/5062592.

[Jasa] Jasper. *TeX StackExchange user profile: Jasper*. https://tex.stackexchange.com/users/319072/jasper. Accessed: 2025-08-12.

[Jasb] Jasper. *TikZ Parametric Surface Debugging Request*. TeX StackExchange. Asked January 10, 2025; accessed August 6, 2025. URL: https://tex.stackexchange.com/q/734691.

[Nyb11] Daniel Nyberg. *Analysis of Two Common Hidden Surface Removal Algorithms, Painter's Algorithm & Z-Buffering*. http://www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2011/rapport/nyberg_daniel_K11061.pdf. Independent thesis Advanced level (professional degree), 10 credits / 15 HE credits. Supervisors: Lars Kjelldahl. Examiners: Mads Dam. 2011.

[RA76] David F. Rogers and J. Alan Adams. *Mathematical Elements for Computer Graphics*. First edition. New York: McGraw-Hill, 1976, p. 239. ISBN: 0070535272.

[Ski25] Skillmon. *Answer to "3d point sorting in tikz"*. Accepted answer with LaTeX3-based segment list and sorting macros. 2025. URL: https://tex.stackexchange.com/a/734098.