The Investigation of Graphics in the Processing Language (MIT Media Lab) and the Development of Applets Involving Advanced Concepts

John Trent
TJHSST Computer Systems Laboratory
2005-2006

Abstract

This project is set out to study graphics through the use of the Processing language (MIT Media Lab). Through research and tutorials, I gathered enough knowledge to create several applications using advanced concepts in graphics. I decided to use Processing to study computer graphics because it is a useful tool in graphics development. It is both user-friendly and easy to use. The Processing language is java-based and all that is required to begin using it is a basic pre-existing understanding of Java. Over the course of the year, I designed multiple applets, each involving a different aspect of 3D graphics. The concepts involved included the implementation of a 3D camera, image masking, implementation of a particle system, lighting effects, time distortion effects, the implementation of a billboard, 2D to 3D conversion, and basic physics.

1 Introduction

i. Purpose

The main purpose of this project is to create a number of applets through the use of Processing that demonstrate use of advanced concepts in graphics. Those interested in graphics and possibly using Processing should be able to use my project as a stepping stone. Processing only requires pre-existing knowledge of Java and use of the on-line reference. The intended goal of this project is to gain a better understanding of 3D graphics through study and research.

ii. Scope of Study

This study covers beginning with Processing, learning the language and progresses all the way to the use of Processing in complex graphics concepts. Several concepts were studied at great length with each applet being primarily designated to a particular one.

2 Background and Research

Processing is an open source programming language that is java-based. Users can code various graphic effects using the sketchbook tool and then export the "sketches" into applets. Processing is an open source program developed by Casey Reas and Ben Fry in the MIT Media Lab. The program, while still in the beta stage, is periodically updated with a discussion board available for solving any technical issues that may arise.

In designing most 3D environments, it is preferable to integrate the concept of a camera. In most computer games played through the first person, the camera is free to roam about the virtual environment. It can explore through moving and turning. In games played through the third person, the camera is apart from the character and can be lead around, in essence filming the action. Most important in terms of this project, one can implement a camera to look at a model from all sides in a 3D graphics program.

Polygons can be moved about in 3D space just by moving their points. Now, one can see how this will help us to implement a camera. After all, once rendering routines take into account the position and orientation of the camera, one can zoom around a 3D scene with only a few lines of code.

Surprisingly, math actually comes in *handy* when dealing with 3D graphics. When concerned with changing the position of a point in 3D space, it merely depends on how it is viewed by the camera. Let's call that point W, for World point. The point that one will obtain as a result of the various operations called C, for Camera point. The camera can be moved forward, backward, left, right, up and down. To handle that,

```
C.x = W.x - cam->x
C.y = W.y - cam->y
C.z = W.z - cam->z
```

When the camera turns left, all the points will seem to rotate right. When the camera turns down, all the points seem to rotate up. And when the camera spirals, all the points seem to spiral the other way.

Since 3D has 3 planes, the rotation must be done three times. The x-y plane is called roll, y-z plane is called pitch, and the x-z plane is called yaw. To understand roll, pitch, and yaw, picture this. When you move your head up and down is called pitch. Put your left ear on your left shoulder and then right ear on your right shoulder that is roll. Move your head horizontally side to side that is yaw.

So after you have subtracted cam->x, cam->y and cam->z from the point (x, y, z), you rotate the results.

Equations to rotate a 3D point:	Multiply the point vector by matrices:		
x'=z*sin(yaw)+x*cos(yaw)	Matrix to rotate around x axis:		
y'=y	cos(a)	-sin(a)	0
z'=z*cos(yaw)-x*sin(yaw)	sin(a)	cos(a)	0
	0	0	1
x"=x'	Matrix to rotate around y axis:		
y"=y'*cos(pitch)-z'*sin(pitch)	cos(a)	0	-sin(a)
z"=y'*sin(pitch)+z'*cos(pitch)	0	1	0
	sin(a)	0	cos(a)
x"'=y"*sin(roll)+x"*cos(roll)	Matrix to rotate around z axis:		
y"'=y"*cos(roll)-x"*sin(roll)	1	0	0
z"'=z"	0	cos(a)	-sin(a)
	0	sin(a)	cos(a)

Here, you only have to calculate the sine and cosine of the roll, pitch and yaw of the camera once per frame. Once you have them, you can use them for all the polygon transformations. Then, it is only a few multiplications and additions per point. That's about as optimized as you can get. So developers of 3D engines normally use the equations presented above for transformation of points.

Visuals can help bolster a presentation. But by transforming an ordinarily flat graphic to one that appears three-dimensional, you can take that visual one step further and make it "pop" from the slide. Such "3D" graphics can give the illusion of depth, of spatial relationships and — with clever use of custom animation — of motion.

Primitives are the basic geometrical shapes used to construct computer graphics scenes and the resulting final images. Each primitive has attributes like size, color, line and width. For two dimensions, examples of primitives include: a line, circle, ellipse, arc, text, polyline, polygon, and spline.

For 3D space, examples of primitives include a cylinder, sphere, cube and cone. 3D viewing is complicated by the fact that computer display devices are only 2D. Projections resolve this issue by transforming 3D objects into 2D displays.

Lighting models also assist with viewing 3D volumes that are transformed on to 2D displays. For example, a 3D red ball looks like a 2D red circle if there are not highlights or shading to indicate depth. The easiest type of light to simulate is "Ambient" light. This lighting model produces a constant illumination on all surfaces of a 3D object, regardless of their orientation.

Directional light refers to the use of a point light source that is located at infinity in the user's world coordinates. Rays for this light source are parallel so the direction of the light is the same for all objects regardless of their position in the scene.

3 Procedure/Development

The project first began with the investigation of Processing through the study of examples provided on the main website and sample code. A helpful tool was the processing message board where fellow programmers posted errors and problems that they ran into along with helpful tips and solutions.

Based on a few of the sample applets provided, I created a rudimentary applet of my own involving a Rubik's cube. This is where I first encountered the use of camera in graphics. The camera system that I used in this case was rather elementary and was controlled with the use of a mouse. By left-clicking, the cube is rotated according to the position of the mouse. Once the mouse is released, the cube re-centers itself so that the dominant side faces outward. By right-clicking, the individual sections of the cube are rotated. Based on the location of the mouse, the row or column of the cube selected rotates.

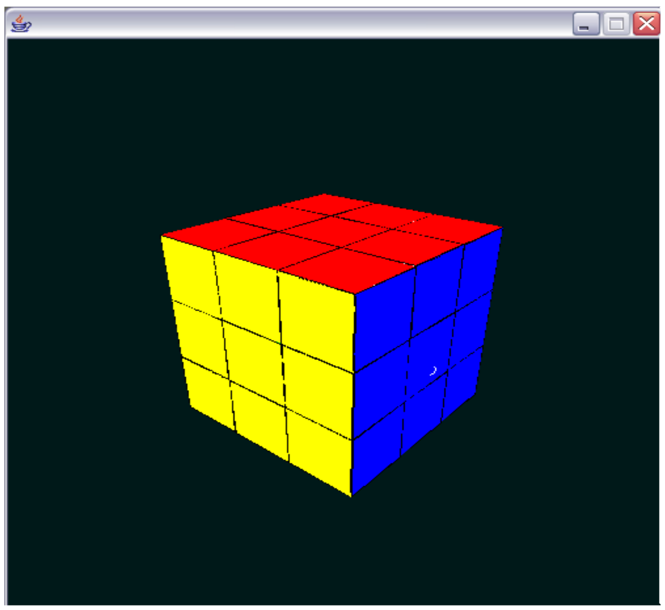
The first original prototype that I developed was a simple applet that created a type of anemone in a 3D environment that could be rotated with the use of the mouse. The anemone consisted of tubes and cones whose size, shape, and direction were randomly generated each time.

The next applet involved the use of a particle system and an initial introduction of a billboard. This applet was where I began to use a more complex camera system which could both be manipulated with the mouse and with the keyboard. The particle system consisted of bubbles. The bubbles implemented image masking to produce the blurring effects. The “w”, “a”, “s”, and “d” keys as well as the mouse. could be used to maneuver the camera. The “f” key turned on and off the blurring effect. The “g” key turned on and off the billboard. A right-click would reset the particle system.

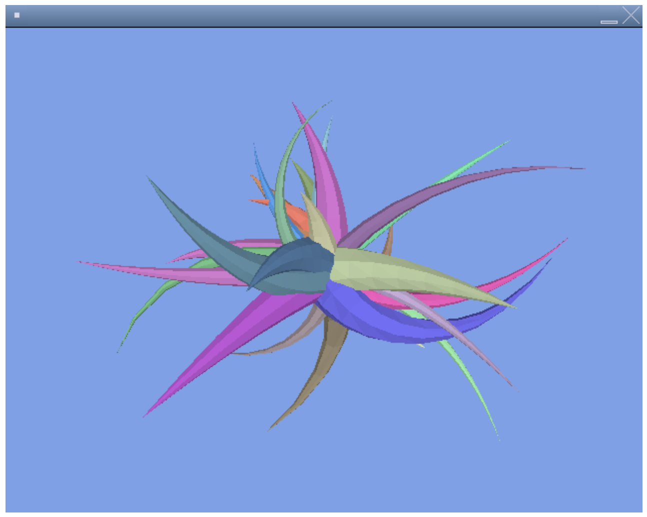
Based on a tutorial, I created an applet involving Jarek's Offset, a means of going from 2d to 3D via recursion. The applet began in the 2D form and involved editing a track for a coaster to travel upon. By moving around nodes, the track would curve and bend accordingly. Once the editor was working, I wrote the camera system to center around the coaster. The coaster itself was rather simple and consisted of 3 blue spheres meant to symbolize a passenger. The camera could be rotated about the rider by means of the mouse.

From my studies of Jarek's offset, I created a second applet involving a submarine and added a panoramic view mode in addition to the 2D and 3D forms. The submarine consisted of a number of 3D polygons. The components were the body, the fuselage, the cabin, and the wings. Similarly to the Coaster applet, it involved editing a track via nodes in the 2D form that was then translated into 3D form through Jarek's offset. The camera could be rotated about the submarine.

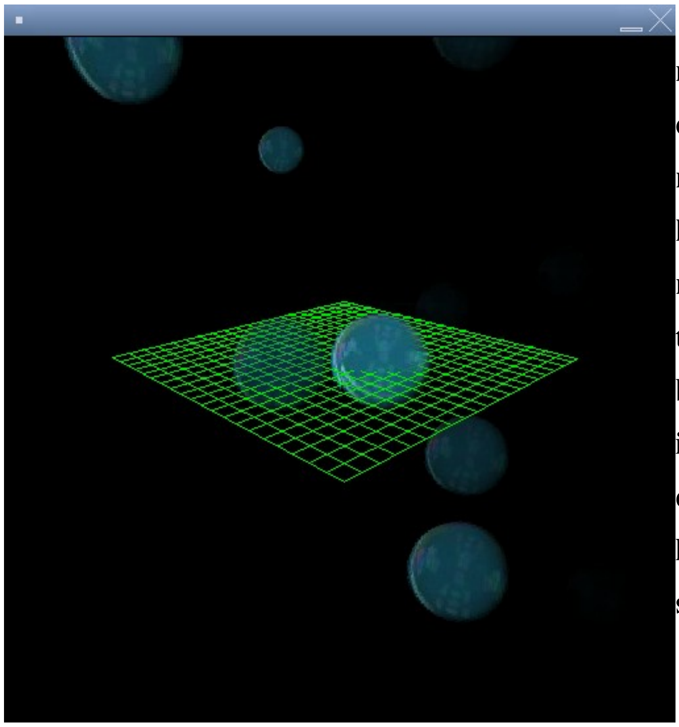
4 Results



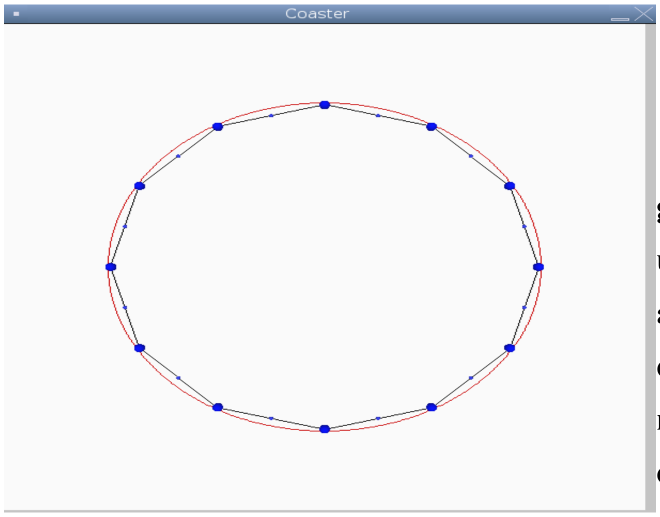
As can be seen from above, the Rubik's ube applet was rather successful. It met all the roperties of an actual Rubik's cube. Each of the ections could be rotated, and each of the six ides functioned properly. The camera controls llow for rotation in all axes. Also, the rotation of he sides is fully animated. The only visual hortcoming of this applet was the discontinuity f the lines that defined the squares when the amera was rotated.



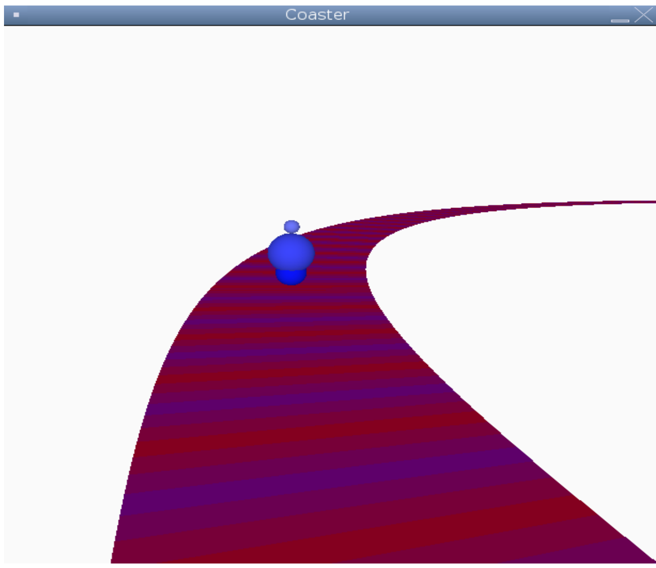
The Anemone applet was perhaps the most visually pleasing. It models the real life sea creature, complete with waving tentacles. It maintained better visual consistency in modeling, the only flaw being that the tentacles would occasionally intersect one another. The same camera system that I developed in the Rubik's cube was used in this applet as well, allowing three axes of freedom.

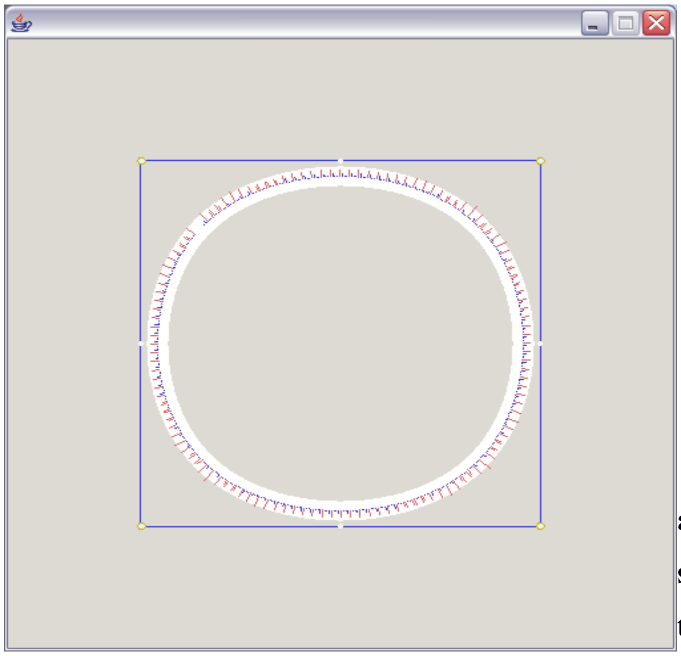


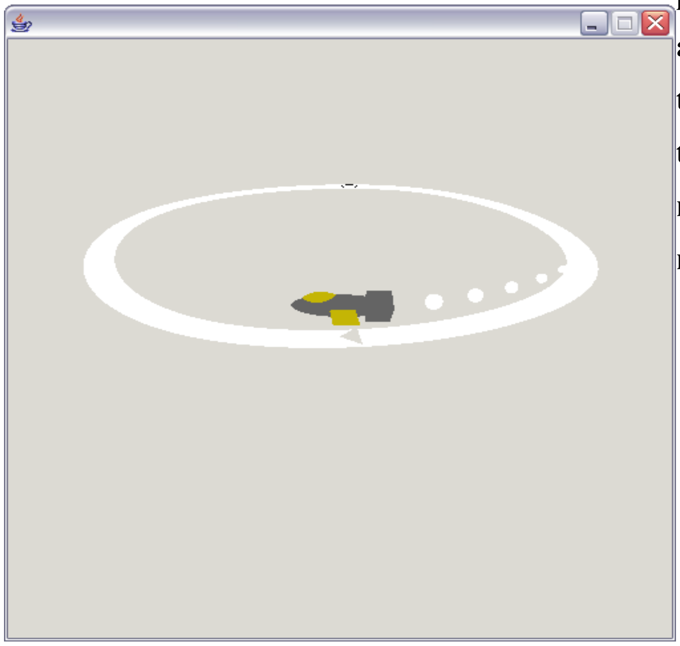
The Bubbles applet involved the greatest number of graphics techniques. I improved the original camera class by adding the ability to move the position of the camera via the keyboard. I implemented a particle system to model the motion of the bubbles, image masking to give the bubbles their transparent effect, and a billboard to enhance the pseudo 3D effect. I also implemented a “slow motion” mode that emulated blur effects similar to that found in the latest 3D games. Overall, this applet was a huge success.



The Coaster applet is feature complete. A graphical mouse-driven track editor allows the user to easily create a track for their roller coaster, and the 3D mode renders the track and runs the coaster. The camera from the Bubbles applet was modified to follow the roller coaster, and a number of graphical effects were included. This applet required skills that I learned from the previous applets, combining elements from Rubik's, Anemone, and Bubbles.







The Submarine applet was the last applet I undertook, and was not yet entirely stable by the end of my project. However, all of the functionality that I planned for was still present. The applet is similar to the Coaster applet, but adds in the ability to build track in the z-axis. The 3D mode is fully working and the submarine can travel in all three axes, but more work still needs to be done to make it more stable and graphically appealing.

5 Discussion

3D graphics involves the creation of three-dimensional virtual worlds within the computer. It's like simulating a movie set in which we describe the "actors" and props in the scene, the placement of lights, the viewpoint of the camera, and for animation, changes (such as movement) in any of the above. 3D illustration uses the same process as 3D animation but simply generates static images.

During this initial phase we meet with the client to discuss design concepts, tradeoffs (for example, greater realism may involve greater cost), and obtain any artwork that may already exist (such as logos). Then we design the characters and sets and create storyboards to plan camera angles and action sequences.

Models are then built for objects that appear in the scene. The time required for this can vary greatly depending on the complexity of the object and the level of detail required in the final images. Creating a 3D version of a logo is generally much faster than creating an organic object such as a dinosaur. In some cases we are able to save the client money by reusing existing props from our digital backlot or by purchasing pre-made models if suitable ones exist.

Once a model's geometry has been built, we need to describe its surfaces. This can be as simple as specifying attributes such as color, shininess, transparency, etc. or more involved such as when we create custom texture maps (digital paintings) or use specialized software for things such as "growing" fur. Often, simple objects can be made to appear quite realistic by mapping stock photographs onto their surface (e.g. brick, wood, wallpaper).

Lighting a virtual 3D set is similar to lighting a real photographic set. We choose the type of lights (such as spot or diffuse), their placement, brightness, color, throw pattern, etc. Unlike real lights, we can also choose things such as whether or not the light casts a shadow and its type (more realistic shadows generally take longer to render) and even have "negative lights" that remove light from a scene. We also have fine control over physical simulations such as lens flare, volumetrics (things like "rays" seen through a smoke-filled room), caustics (things like swirly patterns reflected onto a wall near water surfaces), and radiosity (things like color bleeding seen when a white object is placed next to a brightly colored one).

Animation is the illusion of movement by small increments of change between still frames shown in rapid sequence. In traditional cell animation each frame is drawn by hand. In computer animation we can simply "tell" an object to move from one point to another (known as keyframes) over some period of time and allow the computer to generate all of the frames in between. However a fancy song and dance number requires a lot more keyframes than simply saying "start on this side of the stage and end up on the other". Thus the amount of time it takes to animate something is

usually related to the complexity of the movement. Most any object can be animated including models, cameras and lights. In addition to animating position, many other attributes of a scene can be animated such as the color of an object, the point of focus of a virtual camera lens, or the brightness of a light.

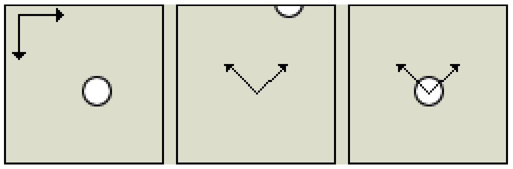
Rendering is the process whereby the description of a scene is turned into an actual image, or for animation, a sequence of images. It is during this process that the resolution of the image is determined (for example, print generally requires vastly greater resolution than video). Depending on a large number of variables, the time it takes to render an image (i.e. a single frame) can vary from a few seconds to days. By carefully controlling factors known to increase render time, test renders can be performed in a fraction of the time of the final render.

Once a scene has been rendered, additional post-processing may be required such as compositing with live action (film or video). Sometimes better control can be achieved by rendering parts of a scene in different passes and later compositing them together. Also some effects can be more easily achieved or rendered faster in 2D than in 3D. Once the final images are assembled the result is output to the medium of choice such as film, video, or CD-ROM.

Unlike other forms of art in which major changes to the appearance can require doing most of the work over, 3D allows us to make some changes quite easily and then simply tell the computer to re-render the scene. Therefore the above "steps" do not necessarily follow the order given.

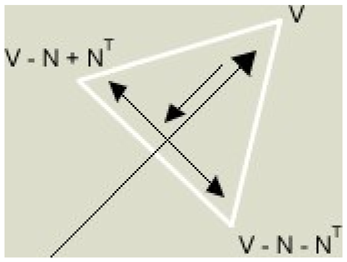
The purpose of the Submarine and Coaster applets was to introduce three dimensional physics into the curve editing system at hand. This naturally requires the addition of a three dimensional viewpoint to the simulation.

To accomplish this, the animation system had to be restructured around a rotation system. This was first tackled in two dimensions, which can be observed by pressing 'w' in the applet. The rotations and positioning was quite simple, as the figure demonstrates.



First, the origin of the window is translated to the center of the screen. Then, about this point, the whole screen was rotated based on a angle calculated using the current and last point in the animation. Finally, the point in focus (white) is translated to the center. This is repeated each draw statement.

The next hurdle is to orient shapes based of the direction of motion. Using merely the current and last animation points the direction of motion can be computed.



The above diagram demonstrates how the triangle and many other directional objects in this program are created. Using the two points a vector (V) is created and subsequently the normal (N). The inverse of this normal along with either the positive or negative transposed normal is added to the current point to achieve the two other vertices needed to create the triangle.

This method was used throughout the rest of the program. All 3D transformations have two main parts, the global transformation which orients the curve and the actors on it into a proper perspective and then local tweaks which reposition specific objects in relation to the master transform.

6 Conclusion

Through the use of Processing, a great deal of tasks can be accomplished in the field of graphics. Hopefully, through the use of my research, others interested in graphics and Processing will be able to use it as starting point and achieve a head start in the field of Processing.

References

1 J. Craig Dunwoody , Mark A. Linton, Tracing interactive 3D graphics programs, Proceedings of the 1990 symposium on Interactive 3D graphics, p.155-163, February 1990, Snowbird, Utah, United States

2 The Khronos Group, "OpenGL ES Overview", Available at http://www.khronos.org/opengles/index.html

3 ARM Ltd., "ARM 3D Graphics Solutions", Available at http://www.arm.com/miscPDFs/1643.pdf

4 Tulika Mitra , Tzi-cker Chiueh, Dynamic 3D graphics workload characterization and the architectural implications, Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture, p.62-71, November 16-18, 1999, Haifa, Israel

5 Michael Pichler , Gerbert Orasche , Keith Andrews , Ed Grossman , Mark McCahill, VRweb: a multi-system VRML viewer, Proceedings of the first symposium on Virtual reality modeling language, p.77-85, December 13-15, 1995, San Diego, California, United States

6 The Mesa Project, "The Mesa 3D Graphics Library", Available at http://www.mesa3d.org

7 **Yonsei University, 3D Graphics Accelerator Group, http://msl.yonsei.ac.kr/3d/**