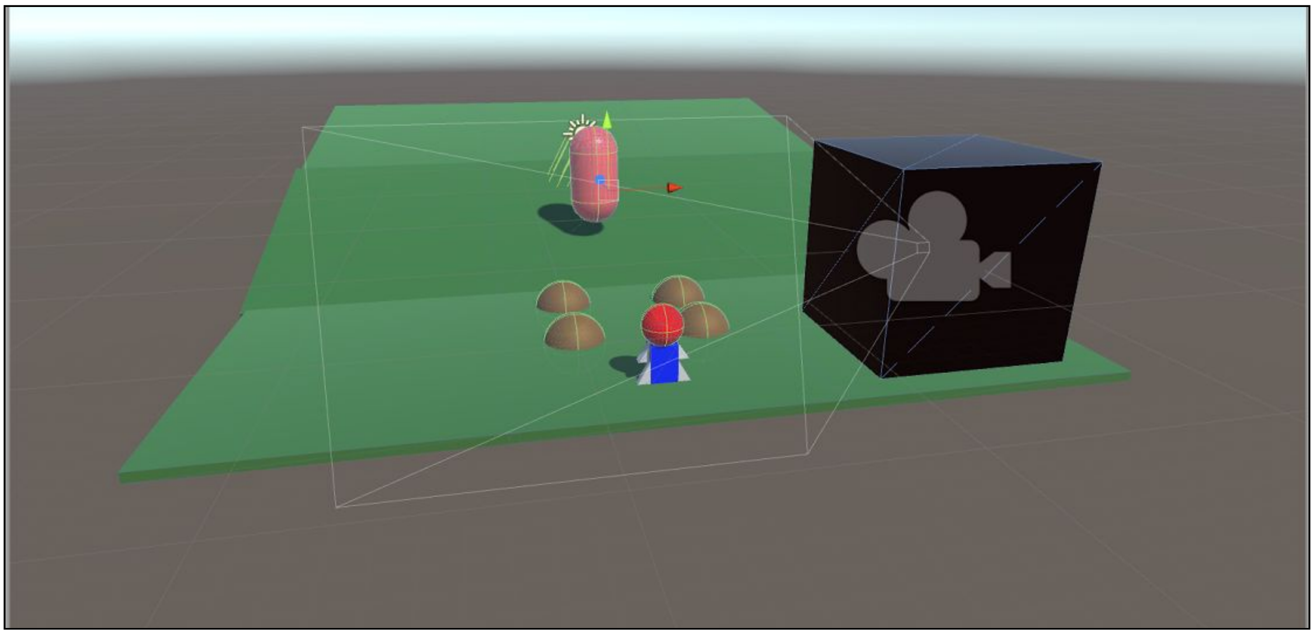
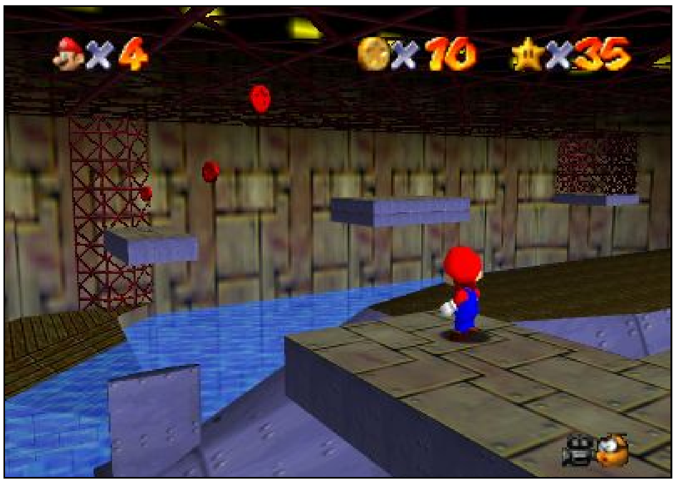


Finding the Optimal Camera Position in 3D Platformer Video Games

by Julian Rice (004914-0045)
Kaohsiung American School (4914)

[1]



[2]

[1] Source: http://s3.vidimg.popscreen.com/original/4/anRVbWJHSVBUblEx_o_super-mario-64---bowser-in-the-fire-sea.jpg
[2] Source: http://1.bp.blogspot.com/-nwX-Ve6ZiQU/VXdTcyO0anI/AAAAAAAAEMU/-u6nBykm44M/s400/dire-dire-docks.png

Table of Contents

Section	**Page**
Introduction	**2**
Gathering the Data	**3**
Formulae	**4**
Exploration	**5 - 15**
Finding Mario’s Screen Ratio	5 - 9
Recreating the Level in Unity	9 - 11
Pinpointing Camera Distance and Angle	12 - 15
Conclusion	**16**
Limitations	**17 - 18**
References	**18**

Introduction

3D platformers have been regarded as one of the rarer genres of video games where those games are either marked as a failure or a masterpiece. With my extreme passion in gaming and video game design, I've decided that with the completion of my IB Diploma, my next step is to select Computer Science as a major and proceed onto video game programming and design post-college. Since my early childhood, I've spent hundreds of hours dedicating myself to 3D platformers - Super Mario 64 being a prime example. Other classic 3D platformers like Super Mario Galaxy, Sonic Adventure, and the Banjo Kazooie series have all contributed to my passionate heart as well. Taking the time to learn how to program and design 3D platformers in High School has also led me to a question to why these classics are so perfect in their level design, character animations, and music score.[3]

Image 1: Super Mario 64 Box Art

With my involvement in various 3D graphics and programming applications such as Unity, and Unreal Engine 4, I've decided to create a math-based exploration towards what would be the optimal camera position in these kinds of 3D platformer games. Unity and Unreal Engine 4 are development platforms that allow for the creation of simple *and* high-end games. They're free to download for use, but programming and graphic design must be learned from an outside source. I will more specifically be using Unity to help recreate a *simplified* version of a level found in Super Mario 64. Unity is a great choice for this exploration because of its capabilities in programming and its simple interface. Every 3D platformer has a camera, which projects the screen of the game that the player sees, and is *absolutely crucial* in making a video game's gameplay smooth and playable. While 3D platformers generally have multiple camera options, I will be primarily focusing on the option that gives you a 3rd person perspective of the character that the player is controlling. There are a plethora of factors to consider in terms of the camera position, mainly due to the size of the character in relationship to the level's scenery and surroundings and the camera distance from the character. This will be done by specifically looking at the camera used in the critically acclaimed classic, Super Mario 64, creating a similar situation with camera placement through the 3D programming engine, Unity, determining the character's distance from the camera, and using vectors to determine the optimal position for the camera. I've chosen Super Mario 64 because it has been regarded as the paradigm of 3D platformer games, receiving an average score of *above* nine out of ten. Almost all critically acclaimed 3D platformer games have a near perfect camera position, and my urge to develop video games will increase after finding the result to where the most optimal position for camera placement is. As of November 2013, over 1.2 billion people around the world play video games as well[4], meaning that finding the answer to this fascinating question could affect over a billion gamers, providing an even more pleasant and enjoyable game experience for players around the world.

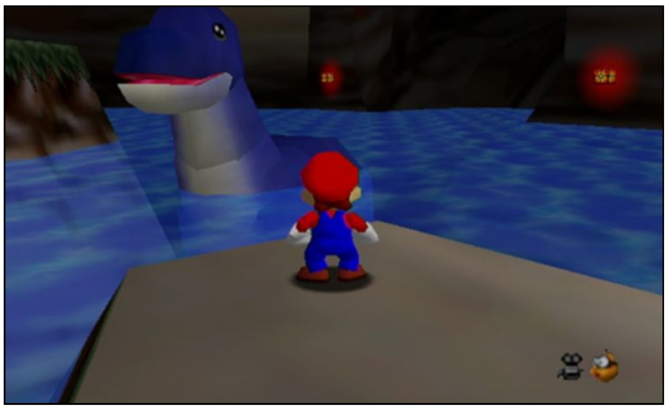


Image 2: Hazy Maze Cave, a level that takes place inside of a cave. From Super Mario 64.

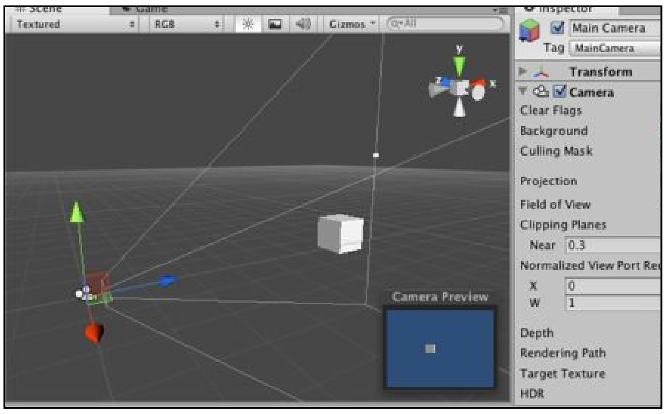


Image 3: The game programming engine, Unity. I will create a simple level using this application.

[3] Image Source: http://www.mariowiki.com/File:Super_Mario_64_Boxart.png
[4] Source: http://venturebeat.com/2013/11/25/more-than-1-2-billion-people-are-playing-games/

Gathering the Data

My first goal in gathering sufficient data is to find the distance of the camera to the playable character, Mario. I will do this by obtaining a frame/image of a level from Super Mario 64, creating a document that will allow me to find Mario's and the screen's area, then finally finding the ratio between the Mario's area and the screen area. After the first step has been completed, the level used in the screenshot will be recreated in Unity, allowing more access to perspectives and the ability to find the points and vectors that will *then* allow us to find the distance between the camera and Mario. The recreated level will be simplified for the sake of saving time and ensuring efficiency in finding optimal camera position. Unity will also be used to help determine Mario's location (as *xyz* coordinates). The camera's location will be set to the origin, (0, 0, 0). The third step has to do with finding the actual distance and angle of the camera used in the simplified version of the level. I will use Mario's head and the camera's location to find the angle between the vector *connecting* Mario and the camera and the *y* axis. I will also need to use the intersection point of two vectors to verify my answer. The assumption is made that the optimal camera location in Super Mario 64 will be identical to the camera location found through the recreated Unity level.

One of the *key* things to take note of is that the optimal camera position *is not* the coordinates where the camera is located. Instead, it consists of the distance *and* angle from camera to the main character. We will assume that our recreated Mario and camera will lie on the same *x* (0, in this case) on the *x* axis, meaning that only *two dimensions* (*y* and *z*) have to be considered when working. The question is more specifically, "Where is the optimal camera position located *in relation* to Mario?" In a game like Super Mario 64, the camera will constantly follow Mario as the player controls him, meaning that there really isn't any exact point in space that can be considered *optimal*. The optimal point that the camera would remain at would be constantly changing as the Mario moves. However, the standard settings in Super Mario 64 are set, meaning that the distance and angle from Mario won't change when he's running on a *flat* surface.

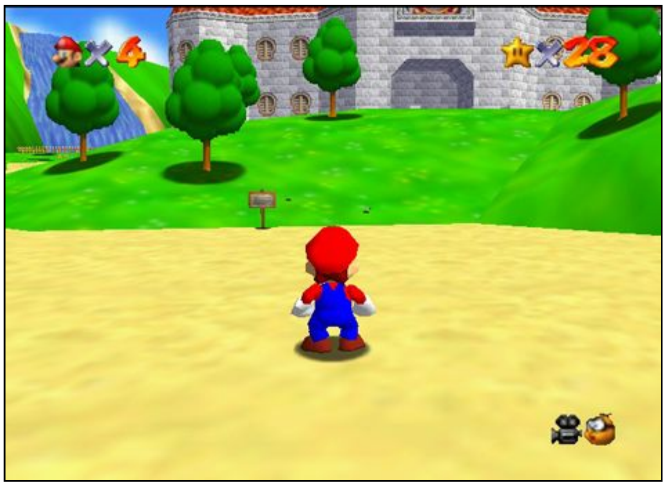


Image 4: An image of Mario in Super Mario 64. The size of his character in relation to the size of the image/screen will be found as a ratio.

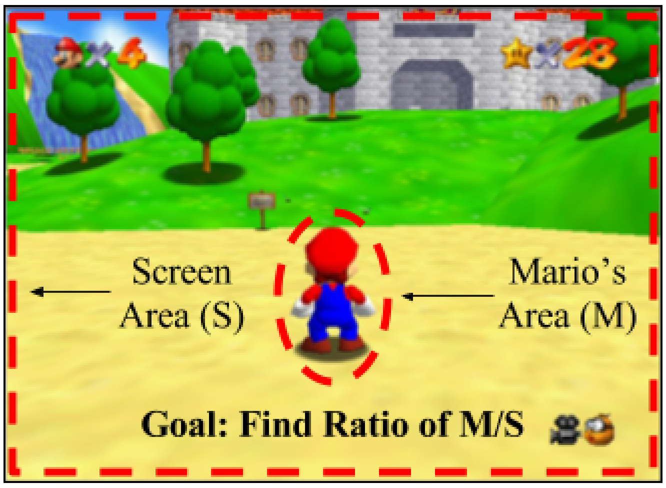


Image 5: Optimal camera distance and angle involves S/M ratio. Finding this ratio will help us come closer to finding optimal distance for this test.

Formulae

Vectors is the primary unit of mathematics this exploration will be focusing on. Because we are analyzing optimal camera distance and angle, the size of the character in relation to the size of the camera is also significant, meaning that a ratio formula will also have to be used. Finding Mario's ratio to the size of the screen will require the use of simple shapes to best estimate his area with efficiency and simplicity. This requires the area formulae for the triangle, rectangle, and circle. Vectors are essentially a quantity that has magnitude (distance), and direction. Vectors are commonly used throughout computer graphics (CGI) and technology, particularly in game and 3D model development. Vectors are also commonly present throughout nature and the world around us. Wind, sea waves, and even shooting basketballs all have vectors used to represent them, as they all have a direction and a magnitude.

$$v \ = \ <(x_2 - x_1),\ (y_2 - y_1), (z_2 - z_1)>$$

Formula 1: Finding a 3D vector from two points.

$$v \bullet w \ = x_1x_2 + y_1y_2 + z_1z_2$$

Formula 2: Dot product of a 3D vector

$$||v|| \ = \ \sqrt{x^2 + y^2 + z^2}$$

Formula 3: Magnitude of a vector

$$cos(\theta) = \frac{x \bullet y}{||x|| \bullet ||y||}$$

Formula 4: Angle between two vectors

$$(x, y, z) + r < a, b, c > \ = \ (u, v, w) + s < e, f, g >$$

Formula 5: Intersection point of two vectors

$$a = l \bullet w$$

Formula 6: Area of a rectangle

$$a = \frac{1}{2}bh$$

Formula 7: Area of a triangle

$$a = \pi r^2$$

Formula 8: Area of a circle

$$ratio = small\ object/large\ object$$

Formula 9: Ratio of an object inside larger object

$$ratio \bullet 100 = percent\ (\%)$$

Formula 10: Percent of an object inside large object

Formulae 1-5 are necessary vector formulae in finding optimal camera distance *and* angle. Again, vectors are basically a point or line that has both magnitude *and* direction. The magnitude represents the value or the distance that the vector is, and the direction represents which way the vector is facing. Direction is more commonly recognized in line vectors, which are connected by two or more points, as line vectors are always going in some direction. Formula 1 is the formula for finding a vector from two points on any plane. Formula 2 is the formula for the dot product, where which is the product of two vectors' individual *x* and *y* coordinates. This formula will more specifically be used for formulas that use dot product in their equation, like Formula 4. Formula 3 is the magnitude of a three dimensional vector, or the *distance/length* (from the initial/start point to the terminal/end point) of the vector. Formula 4 solves the angle between two vectors. This will be used to determine the angle the camera's angle, and is pivotal in finding the optimal camera position, because finding the angle is one factor in optimal camera *location*. Formula 5 is the the formula for the point of intersection of two vectors. This is the formula that will be used at the very end to help verify that the two vectors used have the same intersection point as the point of the camera.

Formulae 6-8 are all equations that find area for different shapes. Mathematical area is the measure of how much space is present on a flat surface, and can be applied to a variety of 2D and 3D shapes. Formula 6 requires the *length* and the *width* of the rectangle. We'll be using the 2D formula in this case because the screen that the player views is a 2D plane, even if the world emulated on the computer is in a three dimensional space. Formula 7 requires the *base* and *height* of a triangle. The triangles will be integrated as substitutes for Mario's limbs (with the *Law of Symmetry* in use). Formula 8 is the area of the circle, requiring only the *radius*. This formula will be used for calculating the area of Mario's head in the ratio portion of this exploration.

Formulae 9-10 are simple formulae that will be used after the geometric equations have been used to find Mario and the screens' area. Formula 9 is the formula that will be used for calculating the ratio of Mario's head in regards to the screen area. Mario is the *small object*, while the screen area is the *large object*. Formula 10 is the formula that will be used to find the percentage of the area Mario takes up. This can be done by multiplying the ratio found in Formula 9 by 100.

Exploration

Finding Mario's Screen Ratio

Stage 1:Start

First off, we need to find Mario's screen ratio. By finding the ratio of Mario's area to the screen area, we'll be indicated as to what percent of the screen he will take up, which will gauge how far (through perspective) the camera should be from the Mario. This section of the exploration will consist of three sections, which are described in the *World Table* below. To make this exploration seem more like an adventure to its final boss, the three sections in this part of the exploration will be named stages (game levels), while this part of the exploration will be named the first world to venture and conquer (World 1).

World 1: Area-Finding Grasslands
Stage 1: Explaining reasons behind finding ratio
Stage 2: Paper used for performing exploration
Stage 3: Finding the ratio of Mario's area to the screen's area

Before actually starting to investigate Mario's area in relation to the screen area, we will need to access a website that allows pixel to cm conversion; I used Translatorscafe's website.[5] Because we'll eventually be printing out a copy of Image 4 to use for finding the area ratio, the game's exact resolution must be converted to centimeters to provide accurate size when measuring the area. Super Mario 64 originally came out on the Nintendo 64 in 1996, and Nintendo 64 games run with a 320x240 screen resolution (240i).[6] This means that the screen dimensions are in the shape of a 4:3 (length:width) rectangle. Let us begin by converting the length (320 pixels) and width (240 pixels) to centimeters.

Pixels	**Centimeters**
320 →	8.467 cm
240 →	6.350 cm

The screen's length and width, in centimeters, of Image 4 has now been calculated, meaning that we can now find the ratio between Mario's area and the screen area. *A detail to take note of* would be that Mario will be redrawn on a blank (but with the identical size) rectangle that will serve as the copy of Image 4 in a simpler form. Mario will be redrawn with simpler shapes to help simplify the process of finding his total area while retaining a high level of precision in the results. Finding the area of the screen was my first step in this long process. I did this by multiplying the length of the screen by the width of the screen, using centimeters as my unit of measurement. After finding the screen area, print Stage 2 and follow the steps listed below Table 2. The *Law of Symmetry* is also another step to consider when finding Mario's area. Because his body model is symmetrical, creating simple shapes to find his area means that there will be each be two of the same simple shapes (triangles), for the limbs of his body.

Table 1: Calculating Screen Area

Variables	Formula 6: $a = l \bullet w$
Length: 8.467 cm	$a = l \bullet w$
Width: 6.350 cm	$a = 8.467 \text{ cm} \bullet 6.350 \text{ cm}$
	$a = 53.76545 \text{ cm}^2$

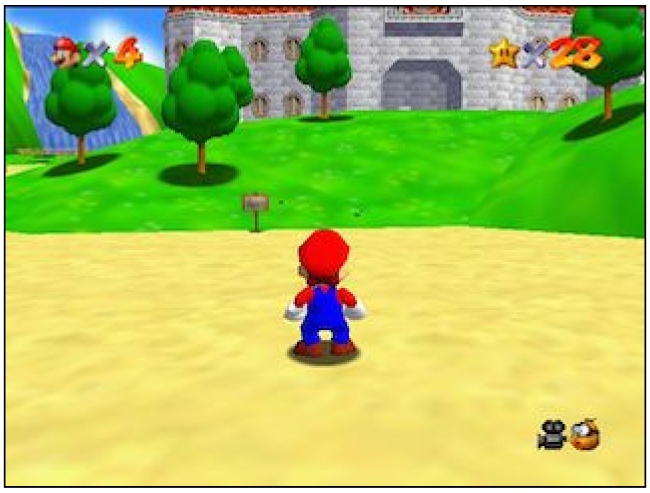
Stage 1:End

[5] Source: http://www.translatorscafe.com/cafe/units-converter/typography/calculator/pixel-(X)-to-centimeter-%5Bcm%5D/

[6] Source: http://nintendo.wikia.com/wiki/Nintendo_64

Stage 2:Start

Paper used for finding the Mario/ScreenArea ratio and percent. Print this piece of paper separately.



Screen Area = **53.76545 cm^2**	Mario Size (x cm) = **cm^2**
Mario/ScreenSize (M/S) = /53.76545	⇒ . *(ratio)*

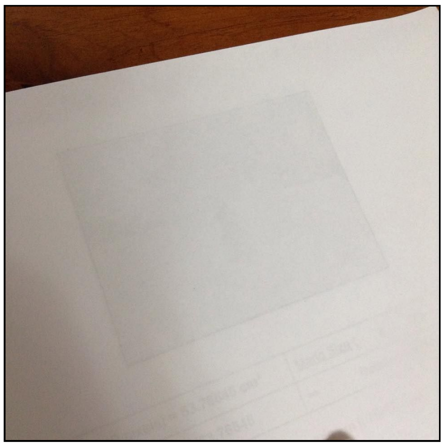
320 pixels = **8.467 cm** \|\| 240 pixels = **6.350 cm** \|\| **53.76545 cm^2**

Stage 2:End

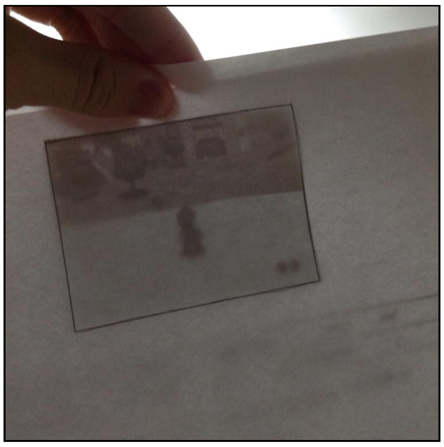
<u>Stage 3</u>:Start

<u>Table 2</u>: Process of Finding Mario's screen ratio.

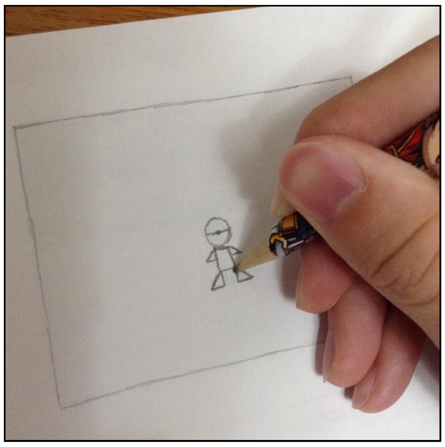
(*Detailed instructions, calculations, and information can be found below the <u>Step</u> images*)



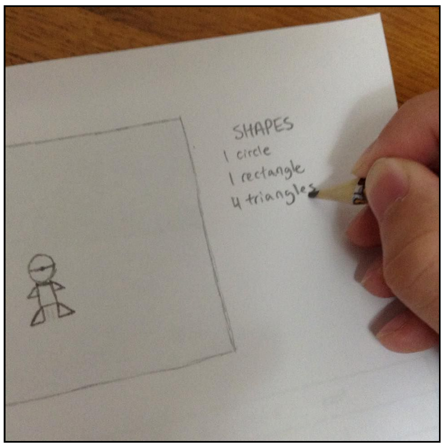
<u>Step 1</u>



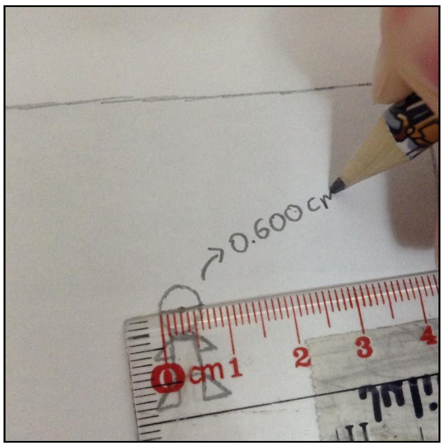
<u>Step 2</u>



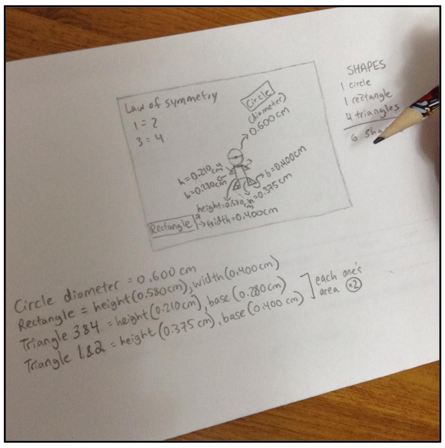
<u>Step 3</u>



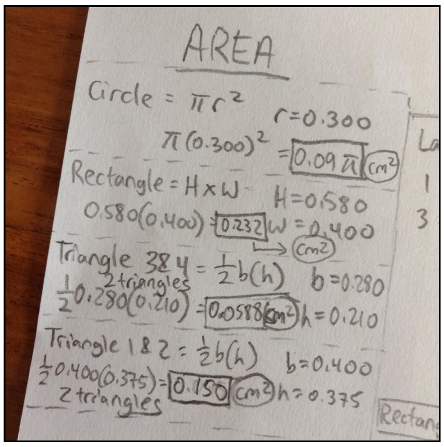
<u>Step 4</u>



<u>Step 5</u>


Law of symmetry
1 = 2
3 = 4
Circle (diameter) 0.600 cm
Rectangle
SHAPES
1 circle
1 rectangle
4 triangles
Circle diameter = 0.600 cm
Rectangle = height (0.580 cm), width (0.400 cm)
Triangle 3&4 = height (0.210 cm), base (0.280 cm)
Triangle 1&2 = height (0.375 cm), base (0.400 cm)
each one's area

<u>Step 6</u>



AREA

Circle = πr^2 $r = 0.300$

$\pi(0.300)^2 = \boxed{0.09\pi}$ cm²

Rectangle = H × W $H = 0.580$

$0.580(0.400) = \boxed{0.232}$ $W = 0.400$ cm2

Triangle 3&4 = $\frac{1}{2}b(h)$ $b = 0.280$

2 triangles

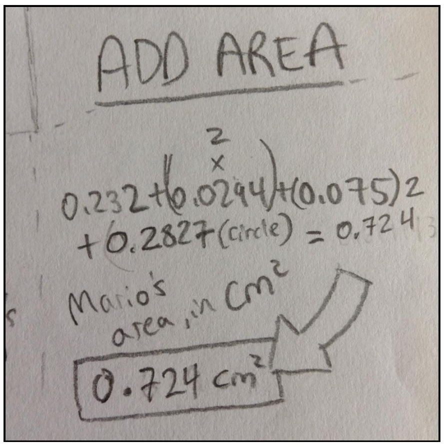
$\frac{1}{2}0.280(0.210) = \boxed{0.0588}$ cm² $h = 0.210$

Triangle 1&2 = $\frac{1}{2}b(h)$ $b = 0.400$

$\frac{1}{2}0.400(0.375) = \boxed{0.150}$ cm² $h = 0.375$

2 triangles

<u>Step 7</u>



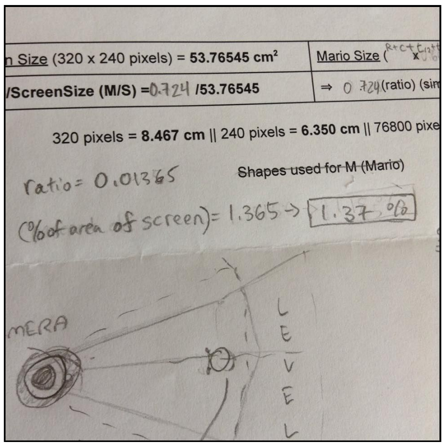
ADD AREA

$0.232 + (0.0294 \overset{2}{\times}) + (0.075)2 + 0.2827(\text{circle}) = 0.724$

Mario's area, in cm²

$\boxed{0.724 \text{ cm}^2}$

<u>Step 8</u>



n Size (320 x 240 pixels) = **53.76545 cm²** | Mario Size (R+C+t

/ScreenSize (M/S) =0.724 /53.76545 | ⇒ 0.724(ratio) (sim

320 pixels = **8.467 cm** || 240 pixels = **6.350 cm** || 76800 pixe

ratio= 0.01365

~~Shapes used for M (Mario)~~

(% of area of screen)= 1.365 → $\boxed{1.37\ \%}$

<u>Step 9</u>

<u>Step 1</u> I first printed <u>Stage</u> <u>2</u> and folded it in half, with both sides facing each other, then proceeded to match the black border from the blank rectangle's side of the paper with the border of the (other side) Super Mario 64 image. Both sides were facing each other.

<u>Step 2</u> I outlined the boundaries of the image using the back of the piece of folded paper. After that, the paper was held in front of light to judge Mario's location.

<u>Step 3</u> I used simple shapes to construct a *simple* version of Mario, using the real Mario from the other piece of the folded paper to help ensure precision in my new and simpler version of him.

Step 4 I took note of which shapes were used to create the *simple* Mario.

Step 5 I used a *cm* ruler to measure the diameter of the circle (head). I continued to measure the bases and heights of the triangles (limbs), and the length and width of the rectangle (body). I measured four unique results and six total results; some weren't unique due to the *Law of Symmetry*.*

Step 6 I then wrote the details I found in Step 5 more clearly.

Step 7 I calculated the area of the circle, four triangles, and rectangle. A table has been created below for a summary of what math was performed for this step.

Table 2: Calculating the Area of Mario's Body Parts

Circle (Head)	**Rectangle (Body)**	**Triangle 1&2 (Legs)**	**Triangle 3&4 (Arms)**
Formula 8	Formula 6	Formula 7	Formula 7
$a = \pi r^2$	$a = l \bullet w$	$a = \frac{1}{2}bh$	$a = \frac{1}{2}bh$
Variables	Variables	Variables	Variables
a = area	a = area	a = area	a = area
r = radius	l = length	b = base	b = base
	w = width	h = height	h = height
Setting Variables	Setting Variables	Setting Variables	Setting Variables
r = 0.300 cm	l = 0.580 cm	b = 0.400 cm	b = 0.280 cm
	w = 0.400 cm	h = 0.375 cm	h = 0.210 cm
Calculations	Calculations	Calculations	Calculations
$a = \pi(0.300)^2$	$a = 0.580 \bullet 0.400$	$a = ½(0.400)(0.375)(2)$	$a = ½(0.280)(0.210)(2)$
$a = 0.09\pi$	$a = 0.232\ \text{cm}^2$	$a = 0.150\ \text{cm}^2$	$a = 0.0588\ \text{cm}^2$
$a = 0.283\ \text{cm}^2$		*Two triangles*	*Two triangles*

Step 8: I then added the different areas I obtained from the circle, four triangles, and rectangle.

Table 3: Calculating Mario's Total Area

a (total) = *circle* + *rectangle* + *triangle 1* + *triangle 2* + *triangle 3* + *triangle 4*

$$a = 0.283\ \text{cm}^2 + 0.232\ \text{cm}^2 + 0.075\ \text{cm}^2 + 0.075\ \text{cm}^2 + 0.0294\ \text{cm}^2 + 0.0294\ \text{cm}^2$$

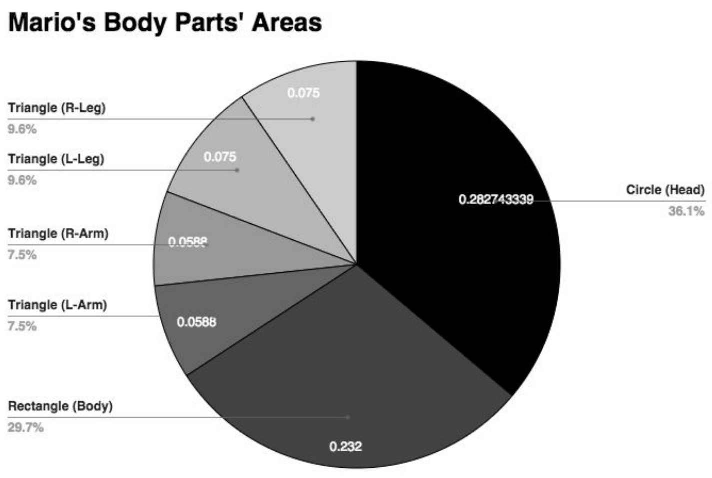
$$a = 0.734\ \text{cm}^2 \rightarrow \text{Mario's Area}$$

Step 9: I found the ratio of Mario to the screen area and turned it into a percentage

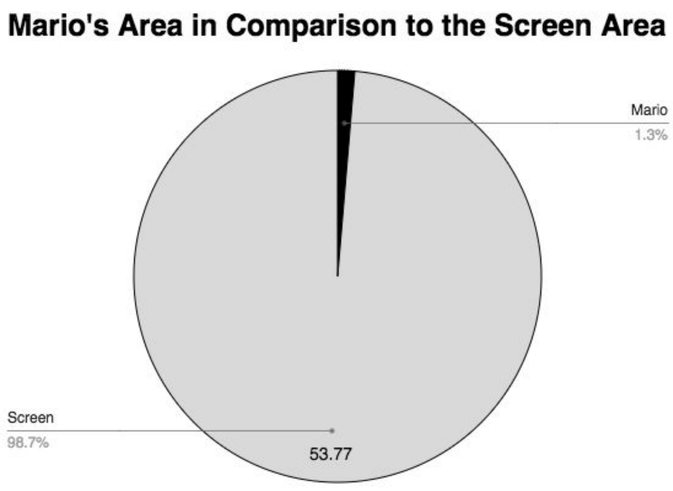
Table 4: Finding the Ratio and Percent of Mario's Area to the Screen Area

Formula 9: *ratio* = *small object*/*large object*	Formula 10: *ratio* • 100 = *percent* (%)
small obj. = Mario's area || *large obj.* = screen area	*ratio* = *smallobject*/*largeobject*
ratio = 0.734/53.765 ⇒ **0.00137**	*percent* = 0.00137 • 100 ⇒ **1.37%**

*The triangles that represented the arms were marked the same, and the triangles that represented the legs were also marked the same, therefore there was a total of six areas. The six account for the area of the circle (one), area of rectangle (two), area of both *arm* triangles (two), and the area of both *leg* triangles (two).



Graph 1: This is a pie chart of the area of the simple shapes that make up Mario. With a total area of 0.734 cm^2, Mario's head has the largest area of all parts of his body. 36% of Mario's total area.



Graph 2: This is a pie chart of Mario's area in relation to the screen area. Notice how Mario only takes 1.37% of the screen, even though he is the primary focus and the main character that the player controls.

I found out that the ratio between Mario's area and the screen area was 0.0137, meaning that Mario's area took up only 1.37% of the screen's area. When talking about this in terms of camera location in 3D platformers, this means that the camera will be at a distance where the character being controlled only takes up a small portion of the screen area. The camera's distance from the character can be determined by recreating the level in Unity. This part of the exploration has helped us get *one step closer* to finding the optimal camera distance and angle because it gives us a general idea of how far away the camera needs to be and the potential direction (alongside with angle) that the camera will face.

Stage 3:End

Recreating the Level in Unity

Stage 1:Start

Although the actual process behind creating a simplified version of the level in Unity isn't *too* related to mathematics, I will explain a few small details regarding the location of the character in relation to the camera. There will be many images with captions describing what's being done in Unity. The goal in this part of the exploration is to help create a basic representation of the stage that we are analysing. Because I have complete control over the level that I create in Unity, I will be able to find exact coordinates, which can then be used to create vectors. This section is more focused on the setup for the comprehensive vector investigation in the third and *final* section of this exploration.

World 2: Precise Icelands

Stage 1: Explaining reasoning behind creation and using Unity

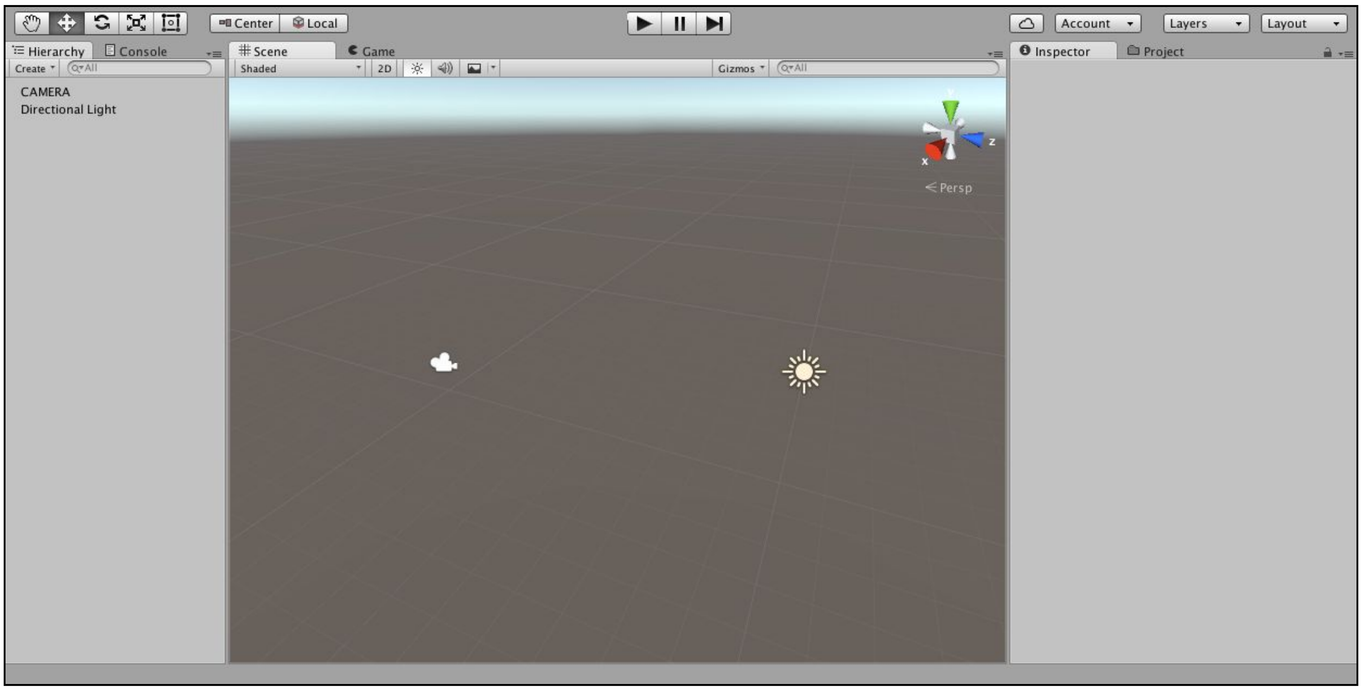
Stage 2: Developing level in Unity with images & descriptions

Even though it has been specified that Unity helps give me more control over the level that I give, it is also a dynamic coding application with a high range of capabilities equipped in its contents. To use *just* a level from a screenshot of a 3D platformer would render the distance from the main character as extremely flawed and very difficult to measure, due to the constraints of the image itself, the inability to rotate the point of view the camera is facing, and the lack of the ability to zoom in with great detail to find the highest accuracy in the results. This is why using Unity to help recreate a *simplified* level will save so much time and will draw the most accurate and precise results as possible.

Stage 1: End

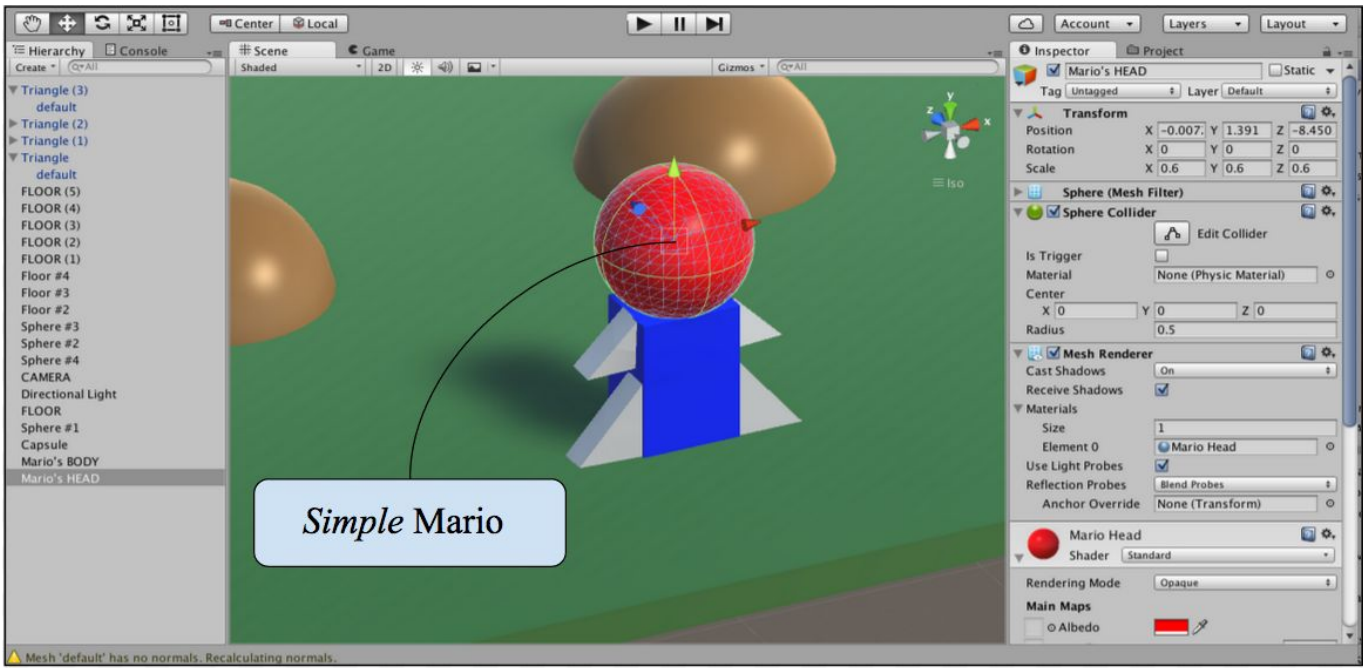
Stage 2:Start

Below, screenshots of the level through Unity can be found. Details can be found *below* each image.



CAMERA
Directional Light

Process 1: This is what the Unity shows after simply creating a new project. I've already adjusted the screen to a 4:3 ratio and have the main camera, or the, "CAMERA," located at the origin. The next step in the process would be creating a somewhat similar, but simpler, level using Unity. At this point, I have done nothing to the project, other than creating it and naming the camera, "CAMERA."

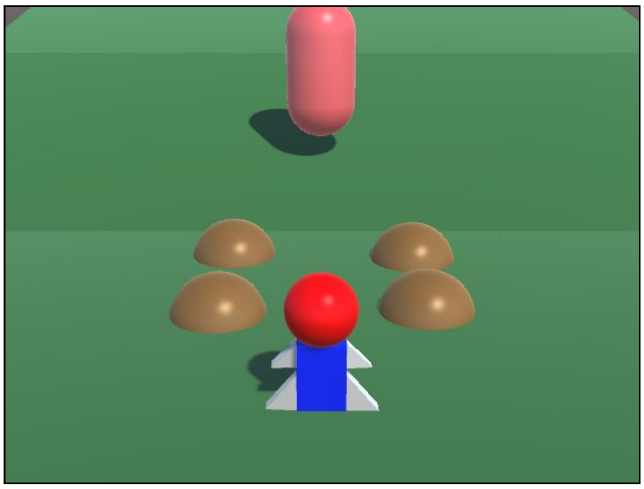
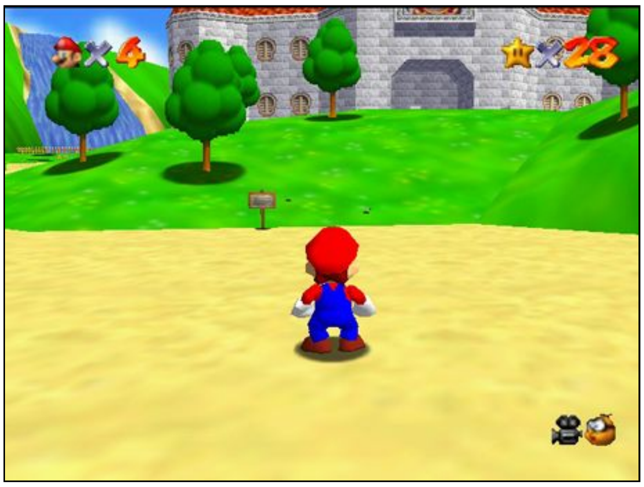


Triangle (3)
default
Triangle (2)
Triangle (1)
Triangle
default
FLOOR (5)
FLOOR (4)
FLOOR (3)
FLOOR (2)
FLOOR (1)
Floor #4
Floor #3
Floor #2
Sphere #3
Sphere #2
Sphere #4
CAMERA
Directional Light
FLOOR
Sphere #1
Capsule
Mario's BODY
Mario's HEAD

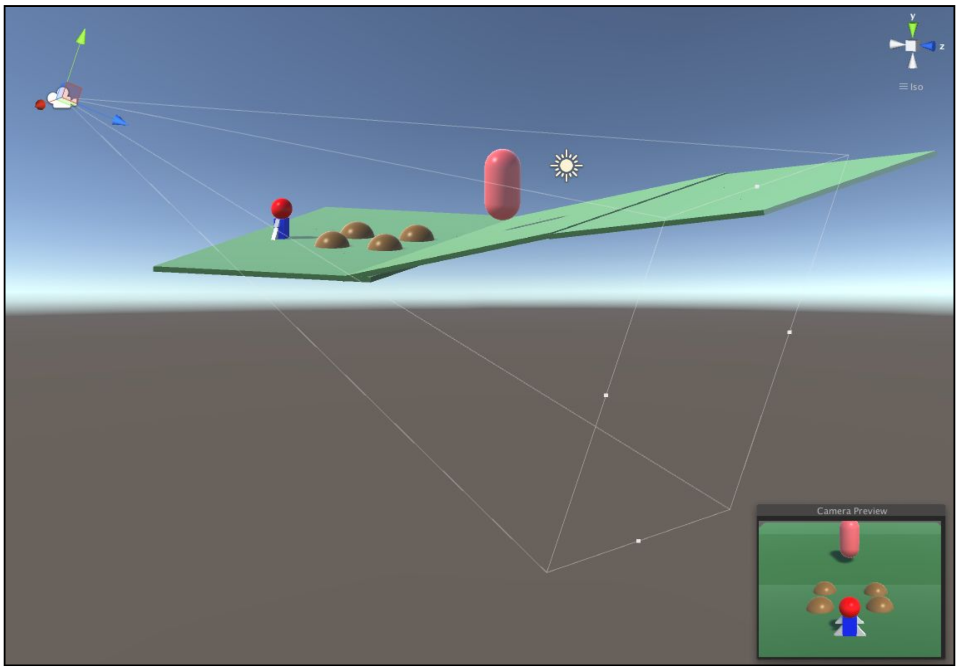
Mario's HEAD   Static
Tag Untagged   Layer Default
Transform
Position X -0.007 Y 1.391 Z -8.450
Rotation X 0 Y 0 Z 0
Scale X 0.6 Y 0.6 Z 0.6
Sphere (Mesh Filter)
Sphere Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Radius 0.5
Mesh Renderer
Cast Shadows On
Receive Shadows
Materials
Size 1
Element 0 Mario Head
Use Light Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Mario Head
Shader Standard
Rendering Mode Opaque
Main Maps
Albedo

Mesh 'default' has no normals. Recalculating normals.

Process 2: I've now taken some time to recreate our new Mario using simple shapes. The sphere highlighted is his head. Take note that while the models in this *and* Super Mario 64 are in three dimensions, the screen is in two dimensions, meaning that one coordinate (x, y, or z) does not need to be considered when meeting the same requirements needed for the ratio of Mario's area and the screen area.

Process 3: This is the final version of the *simple* level, made using Unity. The four brown semi spheres and the red capsule in front of our simplified Mario are all simply small objects that serve as decoration and have nothing to do with the mathematical calculations in the next section. The level has been recreated to the point that the ratio between Mario's area and the screen area are equal to each other. While *Simple* Mario is located a little bit lower on the map… in comparison to Mario in Super Mario 64, they take the same area and percent (ratio) of the screen up. The next step is to show the image of the level and the camera from a distance and unusual location



Process 4: This highlights the camera and its relation to the character in terms of distance and angle. Evidently, the camera preview on the bottom right shows exactly what the camera is projecting onto the screen, meaning that this image above is a great representation of the level we are exploring because it provides us with a completely new perspective of the level. Also, the white lines coming out of the camera symbol are *also* vectors! While these vectors will not be used in this exploration, they are key to take note of, as finding their *vector equations* would be an extraordinary opportunity for finding the perfect intersection points, and eventually, an even *more* precise measurement of the optimal distance and angle from *Simple* Mario to the camera. However, Unity does not reveal the coordinates or any points of these white vectors, hence this limits us in obtaining the ultimate values that dictate the word "optimal."

Stage 2:End

Pinpointing Camera Distance and Angle

Stage 1:Start

After using Unity to create a simple level similar to Image 4, the next step is pinpointing the distance from the camera to *Simple* Mario on the *yz* plane. In a 3D platformer game, the first level you start gameplay at has the main character located, usually standing, on the floor ∴ the *xz* plane will be labeled as the floor. This can be seen below in Image 6. The camera's location has been set *to the origin*, or (0, 0, 0), to help simplify the vector work that will be done this section. After the recreation of the level, *Simple* Mario has been adjusted to the same area ratio as the Mario in Super Mario 64, and is located at point (0, -3.36, 6.52).

World 3: Final Lavaworld

Stage 1: Camera and character's points on the *xyz* plane

Stage 2: The *d* vector running through *Simple* Mario and camera

Stage 3: Magnitude, *y* vector, and finding the camera's angle

Stage 4: Verifying location with intersection point of two vectors

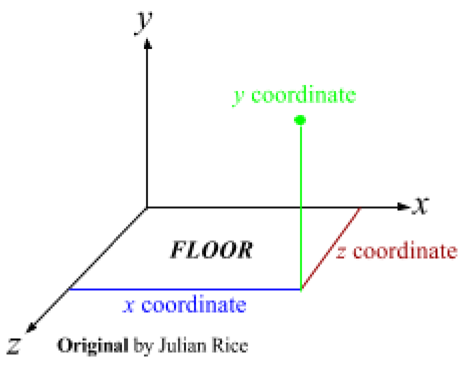
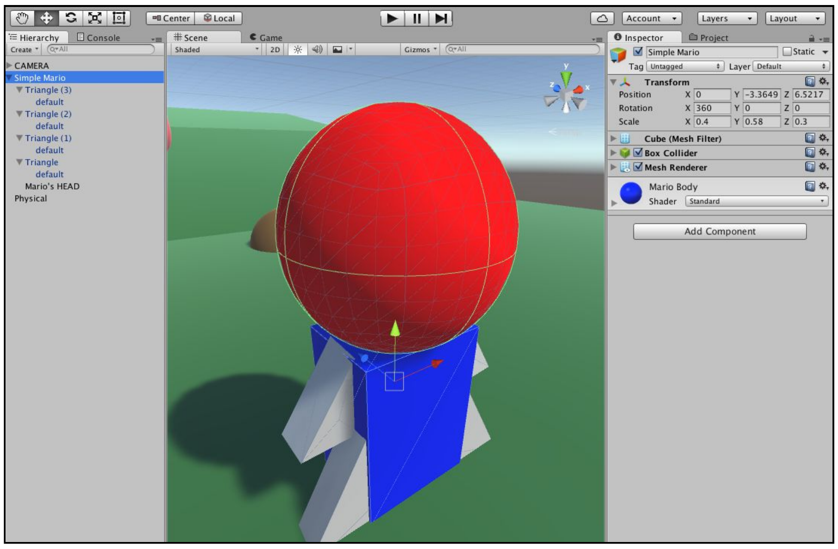


Image 6: The three dimensional plane, with the *xz* plane being the floor of the level in Super Mario 64 and the simple level. The *y* vector will be investigated later when finding *camera angle*.



Simple Mario ☑ · Static ☐ · Tag: Untagged · Layer: Default

Transform
Position X 0 Y -3.3649 Z 6.5217
Rotation X 360 Y 0 Z 0
Scale X 0.4 Y 0.58 Z 0.3

Cube (Mesh Filter)
☑ Box Collider
☑ Mesh Renderer
Mario Body — Shader: Standard

Image 7: *Simple* Mario is located at coordinates (0, -3.36, 6.52)

Stage 1:End

Stage 2:Start

First, we must define the *d* vector and explain the process in finding it. The *d* vector is essentially the vector that runs through both *Simple* Mario and the camera. Image 9 (below) shows a representation of the *d* vector on the *yz* plane. The table below will demonstrate the process in finding the *d* vector, now that we know the location of the camera and *Simple* Mario.

Table 5: Calculating *d* Vector

Formula 1: Finding a vector from two points:	Process
Camera point: (0, 0, 0)	$d = \ < (x_2 - x_1),\ (y_2 - y_1),\ (z_2 - z_1) >$
Mario's location: (0, -3.36, 6.52)*	$d = \ <$ (0 - 0), (-3.36 - 0), (6.52 - 0) $>$
Alternate form: **v** = -3.36j + 6.52k	$d = \ <$ 0, -3.36 , 6.52 $>$

(*Simple* Mario's location was found by selecting his head in the recreated level in Unity. The camera *and Simple* Mario are both on the *x* axis, therefore the two *x* points will be zero.)

Now that we have found the vector running through *Simple* Mario and the camera, we can calculate the magnitude of the vector from him to the camera. The magnitude is another way to say the 'distance'. Afterwards, we will also be defining the *y* vector and will use that to help assist us in finding the angle of the camera facing *Simple* Mario. This stage is important as it not only helps indicate the angle the camera is facing, it also helps reveal the camera's distance from *Simple* Mario. Formulae 2-5 will be used in this stage.

Table 6: Exact Distance From *Simple* Mario to the Camera

Formula 3: Magnitude of a vector	Process
$v = < x, y, z >$ $\mathbf{v} = x\mathrm{i} + y\mathrm{j} + z\mathrm{k}$	$\lVert d \rVert = \sqrt{x^2 + y^2 + z^2}$
$v = < 0, -3.36, 6.52 >$	$\lVert d \rVert = \sqrt{(0)^2 + (-3.36)^2 + (6.52)^2}$
$\mathbf{v} = -3.36\mathrm{j} + 6.52\mathrm{k}$	$\lVert d \rVert = \sqrt{0 + 11.2896 + 42.5104}$
Magnitude of *d* vector: 7.33 units	$\lVert d \rVert = \sqrt{53.80} \Rightarrow 7.33484833$ units

These calculations show that the distance from *Simple* Mario to the camera is 7.33 units, which is the magnitude for *d* vector, with its initial point at the camera and terminal point at *Simple* Mario. This is one of the *main* aspects in finding the optimal camera *location* in 3D platformers. The next step is to define the *y* vector and use it to help find the angle that the camera is facing to ensure optimal angle positioning.

Stage 2:End

Stage 3:Start

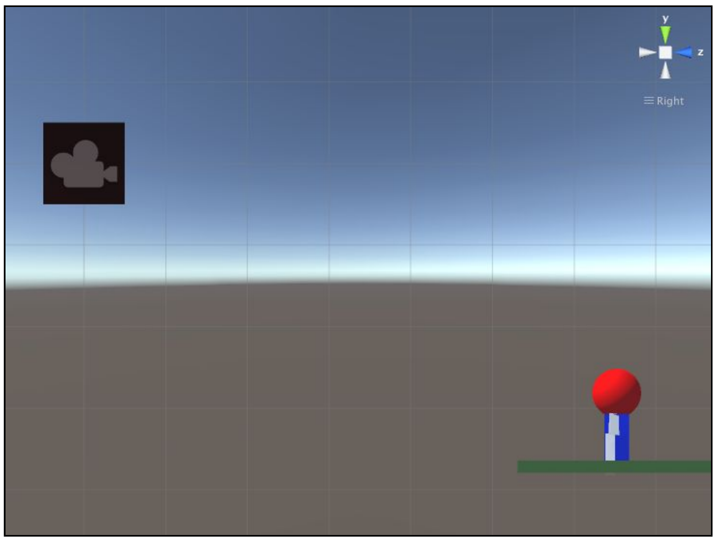


Image 8: This is an image of the recreated scene from Unity, with only the *y* and *z* axes considered. Because Mario and the camera are on the same *x* plane, only two dimensions need to be considered.

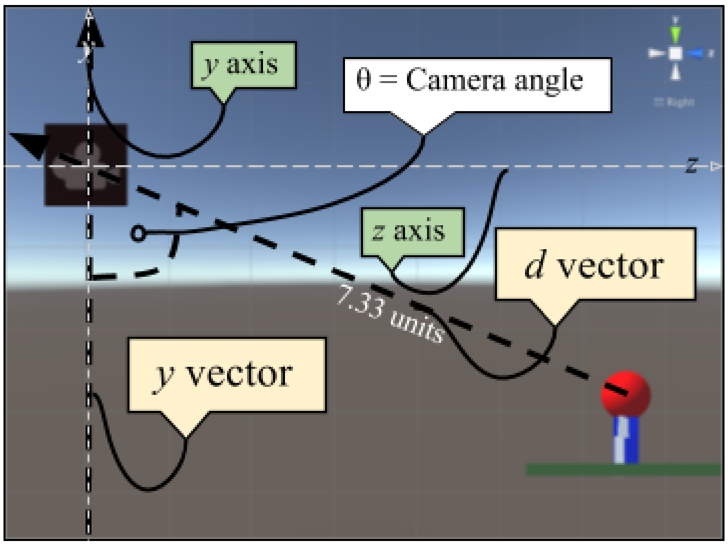


Image 9: The *y* vector will be used to find the angle of *d* vector. This is all possible because we have already calculated *d* vector. All that remains is finding the camera angle from two vectors!

The *y* vector is basically **y** = j, or $< 0, -1, 0 >$. This is because the *y* vector is the vector on the *y* axis and only the *y* axis. We will use this vector and the *d* vector to find the angle that the camera is, facing *Simple* Mario. The *y* vector is negative, or heading towards negative *y* because *Simple* Mario's location is under $y = 0$. If we were to calculate the angle using $< 0, 1, 0 >$, we would have to subtract it by 180º to find its acute angle, or in Image 9, θ.

Table 7: Finding Exact Length of y Vector

Formula 3: Magnitude of a vector	Process
$\lVert y \rVert = \sqrt{x^2 + y^2 + z^2}$	$\lVert y \rVert = \sqrt{0^2 + (-1^2) + 0^2}$
$x = 0 \mid y = -1 \mid z = 0$	$\lVert y \rVert = \sqrt{1} \Rightarrow$ 1 unit

Table 8: Finding the Camera Angle

Formula 2: Dot product of a 3D vector	Formula 4: Angle between two vectors
$d \bullet y = x_1x_2 + y_1y_2 + z_1z_2$	$cos(\theta) = \frac{d \bullet y}{\lVert d \rVert \bullet \lVert y \rVert}$
$x_1 = 0 \mid x_2 = 0 \mid y_1 = -3.36 \mid y_2 = 1 \mid z_1 = 6.52 \mid z_2 = 0$	$d \bullet y = 3.36 \mid \lVert d \rVert = 7.33 \mid \lVert y \rVert = 1$
Process	$cos(\theta) = \frac{3.36}{\lVert 7.33 \rVert \bullet \lVert 1 \rVert}$
$d \bullet y = 0(0) + -3.36(-1) + 6.52(0)$	$cos(\theta) = \frac{3.36}{7.33}$
$d \bullet y = 0 + 3.36 + 0$	$\theta = 62.71672272^{\circ}$
$d \bullet y = 3.36$	$\theta = 62.7^{\circ}$ (Three Sig figs)

Finding out that the angle the camera is at gives us a visual perspective of the screen that the player sees when playing the game. This is the second major necessity in finding the *optimal camera position* in 3D platformer games. The last stage of this last world *and* the *Investigation* part of this exploration will use the most advanced formula included in the Formulae section, Formula 5: the intersection point of two vectors.

Stage 3:End

Stage 4: Start

Finding the intersection point of two vectors is the last step in verifying the data found in Stage 3. Finding the intersection point between the *y* vector and *d* vector will allow us to verify that the camera's location is at the origin. The table below explains the process of finding the intersection point of two vectors. To clarify variables, we will organize such in that *x, y,* and *z* will represent the *d* vector's coordinates, while *a*, *b*, and *c* represent the *xyz* components of the *d* vector. These variables are only applied to this last equation, so do not get confused between this and the previous tables using the variables *x*, *y*, and *z*! This is called the *vector equation*. Variables *u*, *v*, and *w* also act as the *y* vector's coordinates, while the *e*, *f*, and *g* represent the *xyz* components of the *y* vector. One important process to take note of is that when finding the intersection point of two vectors, the two vectors are often times converted from their *vector equation* to *parametric equations*. Parametric equations is basically another form for a vector. The table below may help show the visual difference between vectors in *vector equation* form and *parametric equation* form.

Table 9: Difference Between Vector Equations and Parametric Equations

Vector Equation	Parametric Equation
$(x, y, z) + r < a, b, c >$	$x = x_1 + ra$ $y = y_1 + rb$ $z = z_1 + rc$
Variables explained	Variables explained
$x = x$ coordinate $y = y$ coordinate $z = z$ coordinate	$x_1 = x$ coordinate of a point on the vector $y_1 = y$ coordinate of a point on the vector $z_1 = z$ coordinate of a point on the vector

$a = x$ component of r vector	r = vector variable
$b = y$ component of r vector	r remains as r when writing a vector equation *and*
$c = z$ component of r vector	writing a parametric equation

By understanding parametric equations, finding the intersection point of two vectors becomes a much simpler and more efficient process. The current camera location, as stated before, is the origin, meaning that the answer that we should get from finding the intersection point of two vectors *should be* (0, 0, 0). I will check the answer using d and y_1, which are defined in the table below.

Table 10: Verifying Camera's Location with Calculated Data

Formula 5: Intersection point of two vectors	Process #1 (Converting to Parametric Equations)
$(x, y, z) + d < a, b, c > \ = \ (u, v, w) + y < e, f, g >$	$(0, -3.36, 6.52) + d < 0, -3.36, 6.52 > = (0, 1, 0) + y < 0, 1, 0 >$
Variables	$x = 0 + 0d = 0 + 0y_1$
$x = 0 \mid y = -3.36 \mid z = 6.52$	$y = -3.36 - 3.36d = 1 + 1y_1$
$a = 0 \mid b = -3.36 \mid c = 6.52$	$z = 6.52 + 6.52d = 0 + 0y_1$
$u = 0 \mid v = 1 \mid w = 0$	Process #2 (Finding y and d)
$e = 0 \mid f = 1 \mid g = 0$	$x = 0 = 0$ *so* $\mathbf{x = 0}$
Inserting Variables into Equation	$y = -3.36d = 4.36 + y_1$
$(0, -3.36, 6.52) + d < 0, -3.36, 6.52 >$ is equal to	$y = y_1 = -3.36d - 4.36$ (onto z)
$(0, 1, 0) + y < 0, 1, 0 >$	$z = 6.52d = -6.52$ so $\mathbf{d = -1}$ (back to y)
Calculated Variables	$y_1 = -3.36(-1) - 4.36$
$y_1 = -1 \mid d = -1$	$y_1 = 3.36 - 4.36$ so $\mathbf{y = -1}$
$x = 0 \mid y = 0 \mid z = 0$	Process #3 (Finding the camera's coordinates via d)
Camera's coordinates: (0, 0, 0)	x coordinate $\Rightarrow 0 + 0(-1) \rightarrow 0$
Check (Finding the camera's coordinates via y)	y coordinate $\Rightarrow -3.36 + -3.36(-1) \rightarrow 0$
x coordinate $\Rightarrow 0 + 0(-1) \rightarrow 0$ √ *Checks*	z coordinate $\Rightarrow 6.52 + 6.52(-1) \rightarrow 0$
y coordinate $\Rightarrow 1 + 1(-1) \rightarrow 0$ √ *Checks*	CAMERA'S COORDINATES
z coordinate $\Rightarrow 0 + 0(-1) \rightarrow 0$ √ *Checks*	(0, 0, 0) = Located on the origin

The answer found in Table 10 verifies that the camera is located at the origin, or point (0, 0, 0). By performing this check, we can confirm the following things.

- The intersection between d vector and y vector occurs at the same point as the camera.
- The intersection point and the camera location are both at the origin.

Stage 4:End

Exploration: End

Conclusion

The results found in the third section of the investigation indicate exactly *what* the optimal camera position is for a scene similar to one from Super Mario 64. One of the two aspects of optimal camera position is distance, and we found the distance of *Simple* Mario to the camera by finding the magnitude of d vector. The end result was 7.33 units, which meant that the optimal camera *distance* from the main player, in a 0.0137 screen area ratio, is around 7.33 units. The second aspect of optimal camera position is the angle of the camera facing *Simple* Mario, and we found this by defining y vector and finding the acute angle of d and y vectors. The end result was 62.7º, which meant that the optimal camera *angle* towards the main player in the same screen area ratio is around 37% larger than half of a right angle. These numbers lend insight towards the general use and position of cameras in 3D platformer games, as these cameras are often not looked at in great detail. After this investigation, I have found that there *can* be an optimal camera position in this genre of games, though the limitations behind the results that are found can be very limiting on what can be considered, "optimal." From one frame found in one part of a level in Super Mario 64, I have found a potential optimal camera position that can be used in a similar fashion for future 3D platformers. The use of simple *and* complex math has allowed me to understand the difficulty required to create a work of art that has a camera that operates smoothly and is located in a position that will allow for the best gameplay. This exploration has also opened up a variety of other potential explorations that can be done in the future, with relation to this topic. Some examples of topics that I could actually investigate deeper include researching and analyzing the optimal camera location when Mario jumps onto another block, swims through an ocean, or picks up a wing item and flies in the sky. Most of these other explorations would include more complex math, including related rates, and finding a change in angle of elevation/depression and distance over time. Finishing this exploration also really brings back nostalgic memories of my time with Super Mario 64. I specifically remember playing a lava-filled level, seen in Image 10 for so many hours, looking for potential locations of coins and stars that I hunted. Believe it or not, I found out my solution by adjusting my camera to the standard setting (or, the optimal camera *position*), walking on a grey platform ring, then jumping inside of the volcano that the ring surrounds. Finding out that this standard setup was so useful 10 years later really has taught me that using math in games can *actually* really help you enjoy the game itself way more!

In the end, this investigation has greatly opened up the path for future success in game development, and can be used to create a game that will give the 1.2+ billion gamers around the world a game that will provide them with the greatest experiences imaginable. The money made from selling such a successful game can also serve to help increase your quality of life. Therefore, knowing optimal camera *distance* and *angle* to help define the optimal camera position is only a good thing to know and understand, particularly if the game creators are designing a game similar to Super Mario 64. The use of the optimal camera position found in this investigation *shall* lead the path to new successes in game design in the near future!

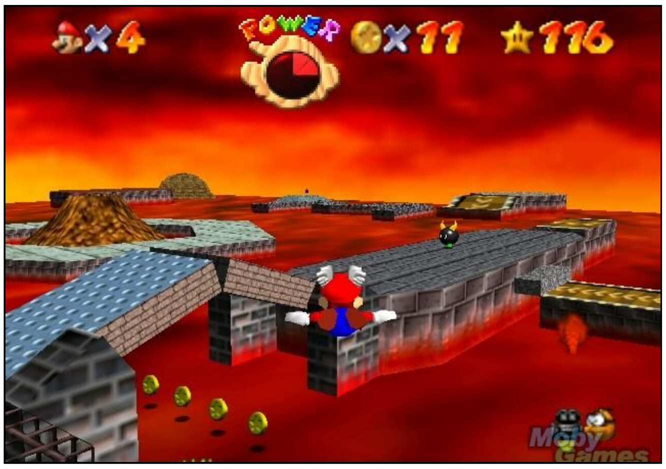


Image 10: An image of Mario flying in a level in Super Mario 64. Notice how the camera angle and distance from Mario is slightly different. I found the star inside the volcano, circled by the grey ring!

Image 11: An image of Mario in the newer 3D platformer, Super Mario Galaxy, where Mario is on a planet. Imagine the camera angle here! I guess we can say that this investigation is *to be continued...*

Limitations

Limitation	Why it was a limitation	What can be done next time
Angle is only optimal when the main character does not jump, fly, or swim.	This is a limit for the exploration because game camera angles are generally varied and modified to follow the character's actions and movements. By not measuring the angles during a jump, when the character swims, or when (in this case) the character obtains an object that allows them to fly, the optimal camera angle is limited and only optimal during the time when the character walks or runs.	If a longer exploration were to be performed, then the camera angle could be measured when actions such as swimming, jumping, and flying occur. Measurements for jumping could be measured with related rates and changing angle of elevation, while swimming would just be a similar type of exploration as this one. Flying may also work using related rates and angles of elevation.
Distance can become shorter than calculated if the player makes Mario run towards the camera.	The camera in 3D platformer games usually stays at a set distance from the character, however, when the player decides to make the character walk or run closer to the camera, the distance naturally becomes shorter. This is a limitation because it changes the camera distance for a few seconds, as it takes time for the camera to return to its optimal distance.	The most interesting part about this limitation is that after a second or two of the character running towards the camera, the camera will naturally adjust itself so that the distance returns to its optimal amount. A deeper exploration may explore the connection between rotation of the camera and its change of distance over time as Mario runs towards the camera (velocity).
Unable to perform vector intersection using two of the four vectors that make up the screen.	Unfortunately, one of the original goals of this exploration was to use the vectors that create what the player can see (screen) to find the camera's location, using the intersection point of vectors failed. There was insufficient data, even in Unity, as points that would help us find what those vectors are were too difficult to find and manipulate. It was also too difficult to manipulate Unity in such a way that an efficient and simple method would be easy to accomplish.	It would have been magnificent to obtain the vectors supporting the screen, and while there was insufficient data in this investigation, there may be possible methods to finding these vectors. One possible method could be creating a 3D model that is located *precisely* where the vector crosses, and using the point where the vector and model intersect to find the vector that runs from the camera to the intersection point. That would be a solution to this limitation.
Optimal distance and angle of camera limited by the 0.0137 ratio.	If the ratio between the main character's area and the screen area was anything significantly (more than ±0.00005) different from 0.0137, then the angle and distance would have to be	This is one of the limits that really hurts the scope of this research paper, and there are nearly no solutions to this problem. The only way to break this limit would be writing an

	different for the camera. This is because the ratio is connected to the distance between the two objects, and can have an effect on the angle that the camera is viewing the character, which all depends on the character's screen area. This is a serious limit that cannot be solved very easily.	extended exploration on this topic, discussing optimal distance and angle from multiple games and multiple levels, with multiple ratios being considered and using them for comparison. Statistics can be performed by comparing different results with different area ratios.
Exploration was based off of and limited to only one frame from just one level from one game.	This limits the scope of the paper and shows that the conclusion reached is rather limited, even in terms of video game design. One frame means that the example used was a *single image* used from *a single level* used from *one* critically acclaimed game. While it shows some legitimacy, being an image from one of the greatest and most paradigmatic games of all time, there still are limits due to the lack of using more frames.	While it would be impossible to create the perfect paper that reveals all possible optimal positions for cameras, increasing the accuracy of the exploration is possible. By increasing the number of frames, most likely by grabbing them from a *large* variety of levels found in the game, the accuracy of the test increases, therefore allowing the analysis to include more viewpoints to discuss.

References

Takahashi, Dean. "More than 1.2 Billion People Are Playing games." VentureBeat. N.p., 25 Nov. 2013. Web. 10 Sept. 2015. <http://venturebeat.com/2013/11/25/more-than-1-2-billion-people-are-playing-games/>

"Convert Pixel (X) To Centimeter." Translatorscafe. N.p., n.d. Web. 07 Oct. 2015. <http://www.translatorscafe.com/cafe/units-converter/typography/calculator/pixel-(X)-to-centimeter>.

"Nintendo 64." Wikia. Nintendo & CC-BY-SA, n.d. Web. 7 Oct. 2015. <http://nintendo.wikia.com/wiki/Nintendo_64>.