Understanding 4-Dimensional Objects Projected into 3-Dimensional Space

Using Light Sources on Rotating Surfaces to Project N-Dimensional Objects

Jean-Dominique Stepek

Department of Computer Science
American River College
04/05/2019

Abstract

Typically, digital images can be viewed as a result of using multiple light sources (*e.g.* LEDs, OLEDs, etc.) to emit a visible wavelength. In computer graphics, a color can be stored in memory as a tuple of red, green, and blue values (RBG) which can be referred to as a pixel. By creating a 2-dimensional matrix of colors, or pixels, a 2-dimensional image can be generated. Thus, to recreate an image of an $\mathbf{n} \times \mathbf{m}$ pixel matrix there must be $\mathbf{n} \times \mathbf{m}$ light sources, if the light source can only represent one color at a moment in time. Representing this pixel matrix with fewer light sources than the product of the number of rows and number of columns will be investigated. The potential of using less light sources has several applications including reducing energy usage, manufacturing cost of a display, as well as subjective aesthetic enhancements.

Contents

Chapter 1

Introduction

Displaying a pixel matrix, an image, using light sources is not a very difficult task in terms of creating an algorithm to write a new color value to each light source. However, this leads to inefficiency in the usage of physical space because a light source that is not activated will still displace the same area as an inactive light source in traditional displays (*e.g.* an inactive television). This is because a light source is stationary and cannot move, but if the light source were able to move faster than a human eye could differentiate between two different positions then a single light source could *appear* to represent several light sources. One method of achieving this task is to create a rotating axle of light sources. By using this method, a single axle can display many of the same pixels that a traditional display can with fewer light sources. This also can reproduce a 'pseudo-holographic' effect even with a single plane of rotating light sources, as long as the axle is thin enough to not obstruct objects in the opposite direction that the light sources face! If additional layers of axles are added, the image will gain a representation in Z-space, or image depth. If the axle is wrapped into a circle then rotated around an axis that passes through the two antipodal points on the circle, a spherical hologram can be produced.

Chapter 2

Mathematics

2.1 Using Rotating Axles to Draw an Image

Imagine there are an n amount of light sources γ expressed as circles, each with radius r, spaced δ units apart, and holding an RGBA value, on an axle. If the axle rotates and each γ_i changes color fast enough, the human eye will see a holographic circle which appears to be an image as opposed to just an ordinary rotating axle (*i.e.* no flicker). The proper angular speed to achieve this effect is anything above 45 Hz, 2,700 rpm, or ∼283 rads/sec [1]. The difficult part of creating the image is to know when γ_i should change color and to which color should γ_i change. To do this, the length L of the axle must be derived using the following formula:

$$L = 2rn + \delta(n-1)$$

This allows a function Γ_2 to generate a Cartesian coordinate of γ_i to be expressed, where Θ is the beginning angle offset of the axle:

$$\Gamma_2(i,\theta) = (i(2r+\delta) - \frac{L}{2} + r)\begin{bmatrix}\cos(\Theta+\theta)\\ \sin(\Theta+\theta)\end{bmatrix}$$

That function can be used to find the Cartesian coordinate of γ_i at time t seconds which means nothing more than $\Gamma_2(i, t\cdot\omega)$, assuming the axle is moving at ω rads/sec.

Finally, a pixel value can be assigned to γ_i from the pixel array P of an image, size $j \times k$ [1]. This is done by using the following function, assuming P is a 1-dimensional array and starts from the top-left of the image.

[1] As $n \to \frac{\max(j,k)}{2}$, the closer the axle gets to representing the image exactly.

$$\gamma_i(t) = P[k(\lfloor\frac{j}{2}\rfloor - \alpha_{21}) + \lfloor\frac{j}{2}\rfloor - \alpha_{11}], \alpha = \Gamma_2(i, t \cdot \omega)$$

However, it's important to notice that the entirety of the expression within the indexing of P will be recalculated every $\frac{2\pi}{\omega}$ seconds. Therefore, for more efficient calculations it would be best to only calculate the value of $\gamma_i(t)$ in the range $[0, \frac{2\pi}{\omega})$ seconds and store those values. Then, when indexing P, use the pre-calculated indices.

Additionally, if more axles are added along the z-axis, an image with a depth dimension can be produced; however, this pseudo-3D image has a limited field of view. In the next section, 3-dimensional images that can be viewed from any angle will be explored.

2.2 Using Rotating Circles to Produce Stereographic Projections

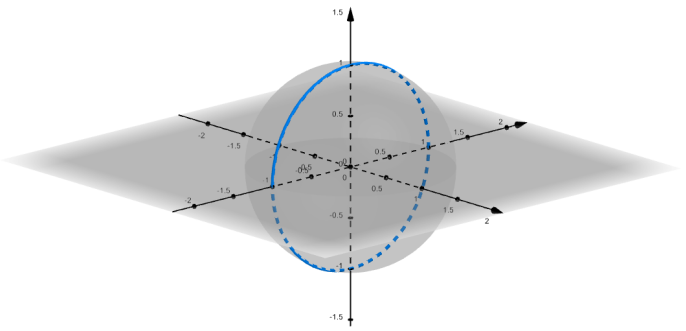

Figure 2.1: A Circle Centered at the Origin

Now, imagine a scenario, similar to the previous section, where there are an n amount of γ, radius r and spaced δ units apart, on a axle, except the axle is now bent into a circle and the circle rotates about the z-axis [Figure 2.1]. Again, if the circle rotates at 45Hz[1], then a flat image can be projected

onto the holographic sphere which allows the image to be viewed from any angle.

The first step in producing the projection is to solve for the radius R of the circle. Before that can be done, the length of the axle must be established. Since the axle is now wrapped into a circle, it's important to have an additional δ added into the original formula for L, thus giving:

$$L = n(2r + \delta)$$

Using the circumference of a circle $C = 2\pi R$, solving for R, and substituting C for the axle length gives:

$$R = \frac{L}{2\pi}$$

This allows for the derivation of the Cartesian coordinate $\hat{d}$ for γ_i, where Θ is an arbitrary angle offset:

$$\Gamma_3(i, \theta) = R \begin{bmatrix} \sin\varphi\cos(\Theta + \theta) \\ \sin\varphi\sin(\Theta + \theta) \\ \cos\varphi \end{bmatrix}, \varphi = \frac{i(\delta + 2r) + r}{R}$$

Using this function, the Cartesian coordinate of each γ_i at time t seconds is the same as the previous section's, where the circle rotates at ω rads/sec: $\Gamma_3(i, t \cdot \omega)$!

Finally, an image be wrapped around a sphere using UV mapping. To achieve this, change the color of γ_i at t seconds through using it's Cartesian coordinate $\hat{d}$ to index a pixel array P of an image, size $j \times k$. Note: It may be necessary to re-map u and v to the image size. This is done with: $u' = k(\frac{u}{\pi} + \frac{1}{2})$, $v' = j\frac{v}{\pi}$.

$$\gamma_i(t) = P[k(\lfloor\frac{j}{2}\rfloor - v) + \lfloor\frac{j}{2}\rfloor - u], u = \tan^{-1}\frac{d_x}{d_y}, v = \cos^{-1}\frac{d_z}{R}, \hat{d} = \Gamma_3(i, t \cdot \omega)$$

After understanding how an image can be projected onto a rotating circle, the next step is to introduce a 3-dimensional object with depth. To be able to achieve this, simply start with a a circle of any radius and add additional circles with differing radii and have all of them spin about a singular axis.[2] However, the angle of each circle should be at an offset to avoid a circle overlapping another (assuming all circles are rotating at the same ω).

[2]As the number of circles goes to infinity, the closer it will be able to model straight lines.

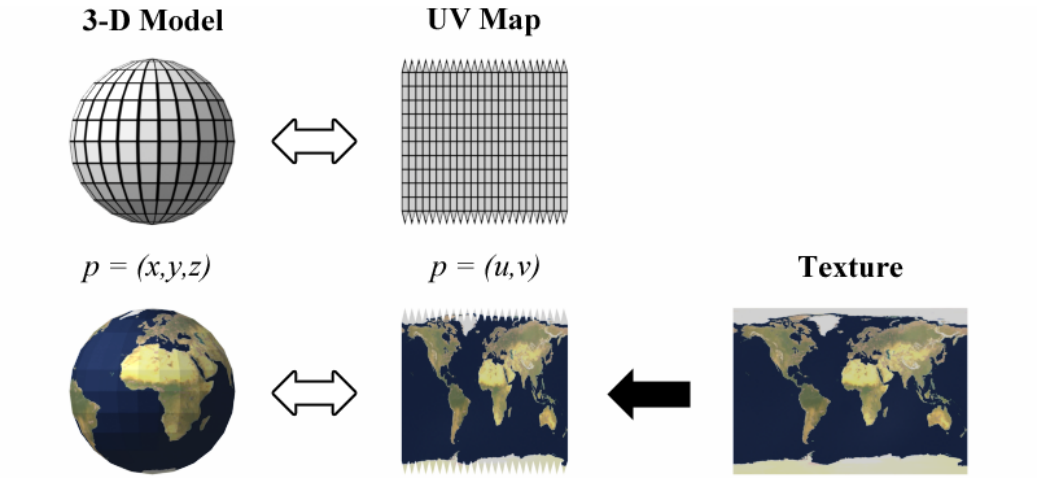

Figure 2.2: Projection of an Image onto a Sphere using UV Mapping

2.3 Understanding 4-Dimensional Geometry With Holographic Spheres

Chapter 3

Engineering

3.1 Rotating an Axle Using DC Motors

3.2 Best Choice for Light Source

3.2.1 Daisy Chaining Neo-Pixels in a Parallel Circuit

3.3 Using Capacitors to Synchronize Angular Velocity

3.4 Materials For Blades

3.4.1 Polycarbonate

3.4.2 Graphene

3.5 Example Wiring Diagram

3.6 Choosing A Micro-controller

3.7 Optional: Integrating IR Sensors for Touch Control

Chapter 4

Programming

4.1 Arduino Studio

4.2 Synchronizing the Rotation Using Magnets

4.3 Digital Hologram

Before creating a physical device that can spin an axle of light sources, a digital equivalent was coded in Processing, a Java framework. It provides necessary information on how to achieve the most efficient and aesthetic effect without using unnecessary resources. Additionally, it allows for the opportunity to visualize the effect of what a physical representation of a line of spinning light sources γ with radius R might generate by using several input parameters:

- L - Axle Length
- C - Axle Count
- η - Light Source Count
- ω - Angles Per Draw (updates/revolution)
- Θ - Angles Per Light Source (angles of γ)

- δ - Offset (spacing between each γ)

By using these variables, a formula for the radius of each γ can be derived:

$$L = 2R\eta + \delta(\eta - 1) \Rightarrow R = \frac{L - \delta(\eta - 1)}{2\eta}$$

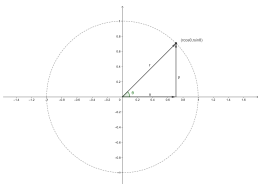

Figure 4.1: Basic Polar to Cartesian Conversion

For ease of writing, the following sets are introduced:

- $T = \{x | x < (C - 1) \wedge x \in \mathbb{Z}^+\}$
- $U = \{x | x < (\eta - 1) \wedge x \in \mathbb{Z}^+\}$
- $P = \{x | x < (\omega - 1) \wedge x \in \mathbb{Z}^+\}$
- $Q = \{x | x < (\Theta - 1) \wedge x \in \mathbb{Z}^+\}$

The next step is to create two sets: one for all magnitudes and one for all angles θ for each γ so that every pair (r, θ) can be used to generate a Cartesian coordinate in terms of our polar coordinate $(r\cos\theta, r\sin\theta)$, which is nothing more than an xy pair, so that the correct pixel value inside a pixel array can be indexed to create a 'polar' field of pixels. The following functions for each item in the magnitude set Φ and angle set Ψ are derived:

$$\Phi = \{i(2R + \delta) + R - \tfrac{L}{2} | i \in U\}$$
$$\Psi = \{\pi(\tfrac{2j}{\Theta} + \tfrac{i}{C}) | i \in T \wedge j \in Q\}$$

It's important to note the calculation for each element in Ψ: the "$\frac{i}{C}$" may be removed in order to represent and calculated each iteration the Once there is a representation for each r and θ, the set of Cartesian coordinates can be made:

$$W(i, j, k) = \{(\Phi_j \cos(\Psi_{i,k}), \Phi_j \sin(\Psi_{i,k})) | i \in T \wedge j \in U \wedge k \in Q\}$$

To draw each γ, the 'circle(x,y,r)' function in Processing can be used; where x and y represent the *cartesian coordinate* of γ and r represents R.

$$\forall\theta \in P(\forall i \in T(\forall j \in U(\forall k \in Q(\text{circle}(\alpha[i][j][k][0], \alpha[i][j][k][1], R)))))$$

To be the most efficient with processing power, it's important to pre-calculate each cartestian coordinate of γ and store it into a coordinate array of size $C \times \eta \times \Theta$ which can be indexed later when writing a new pixel value to a γ

$$index(c, i, j) : \forall c \in T(\forall i \in \eta(\forall j \in \Theta(i\Theta + jC + c)))$$

After calling the program with parameters:

$$Hologram(L = 540.0, C = 2, \eta = 400, \omega = 900, \Theta = 900, \delta = 0.0)$$

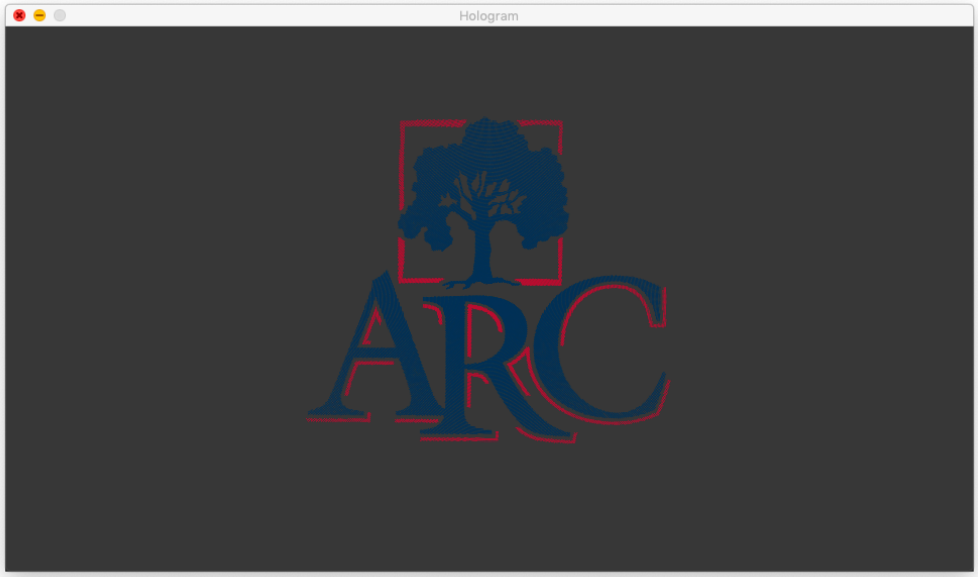

Figure 4.2: American River College Logo

Bibliography

[1] Unios Australia Pty Ltd. *Driving the Flicker-Free Effect.* https://unios.com/wp-content/uploads/2016/06/UN_Driving-the-Flicker-Free-Effect_White-Paper-190205.pdf