Expressing vector V in to a vector V'

1. $Rx\ V' = Rx\ Rx^{-1} V$

$\boxed{V = Rx\ V'}$

$V' = Rx^{-1} V'$

where Rx^{-1} is the inverse of Rx

2. AT diffrence of translation, when doing Rotation, The Inverse of a rotation is the eqivalent of the transposed matrix of the rotation
   a. If $Rx^{-1}(\theta)$ is the inverse of $Rx(\theta)$ then
   $\boxed{Rx^{T}(\theta) = Rx^{-1}(\theta)}$
   b. Transposing a matrix can be easier Than appliying the investion
   c. Example:

$$Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boxed{Rx(\theta)^T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^T$$

$$= \boxed{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

$$\boxed{Ry(\theta)^T} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^T$$

$$= \boxed{\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

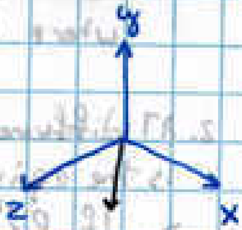
$$\boxed{Rz(\theta)^T} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^T$$

$$= \boxed{\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

Example of Application of coordinate system

1. let assume we have our camera is some position represented by the vectors $\vec{x}'\ \vec{y}'\ \vec{z}'$

2. If we wish to look to the left, we would need to rotate from 90 degrees around the y' vector



3. $\boxed{R_y(\theta)} = \begin{vmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{vmatrix} =$

$$= \boxed{\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}}$$

4. $V'' = R_y(\theta) \cdot V'$

$$V'' = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix}$$

5. $x'' = \cos\theta \cdot x' - \sin\theta \cdot z' = -z'$
$y'' = y'$
$z'' = \sin\theta \cdot x' + \cos\theta \cdot z' = x'$

$x'' = -z'$
$y'' = y'$
$z'' = x'$

6. If the camera coordinate is the same as the global (origin) axis system (x, y, z) (before the rotation), we obtain:

$$x'' = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \qquad y'' = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad z'' = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Assignment 4

Texture Interpolation and Geometric Warping

1. We call "Image Warping" the act of distorcioning a source image into a destination image.

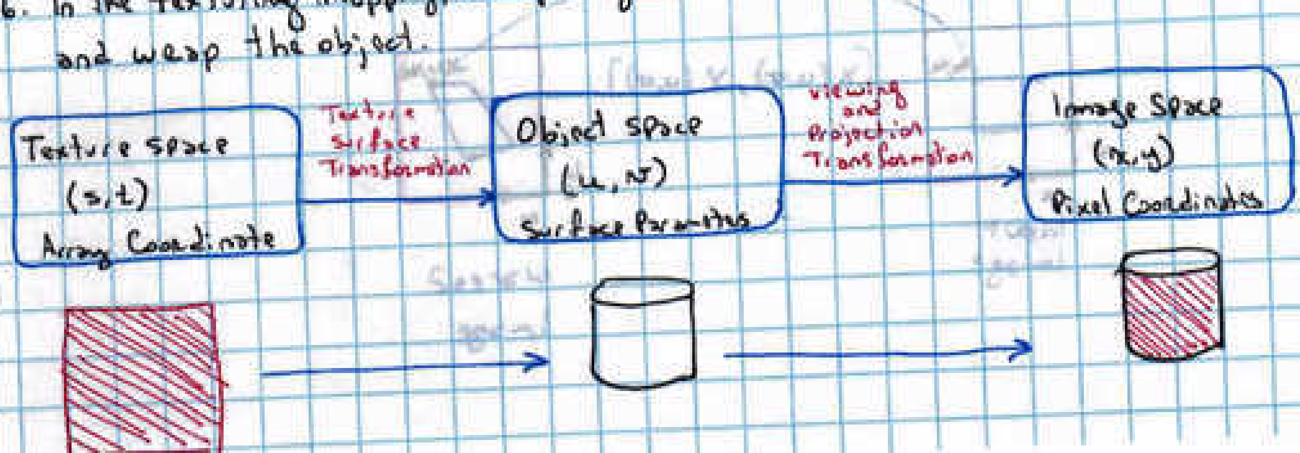
2. We call "Texturie Mapping" when a bitmap or raster image (called "surface texture" is used for adding details to a surface.

3. There are two kind of textures classification:
   a. Explicit Texture Map: This consist of a regular bitmap. A texture coordinates and a subsystem for the texturing processing is required.
   b. Procedural Texture: This would be the program output when computes the texture map. The texture is discomposed in primitive equations and functions. This provide an advantage due the independency in the resolution and the lest repetitive requirement. However, they prove to be high consumers of system resources.

4. Another kind of classification for textures:
   a. Dynamic Maps: computed in real time
   b. Static Map: Created once ~~by using a sequence of bitmaps~~

5. By using a sequence of bitmaps to create a fire effect that can be wrapped around an 3D object we obtain a dynamic and explicit texturing mapping.

6. In the texturing mapping, we specify how the texture map should strech and wrap the object.



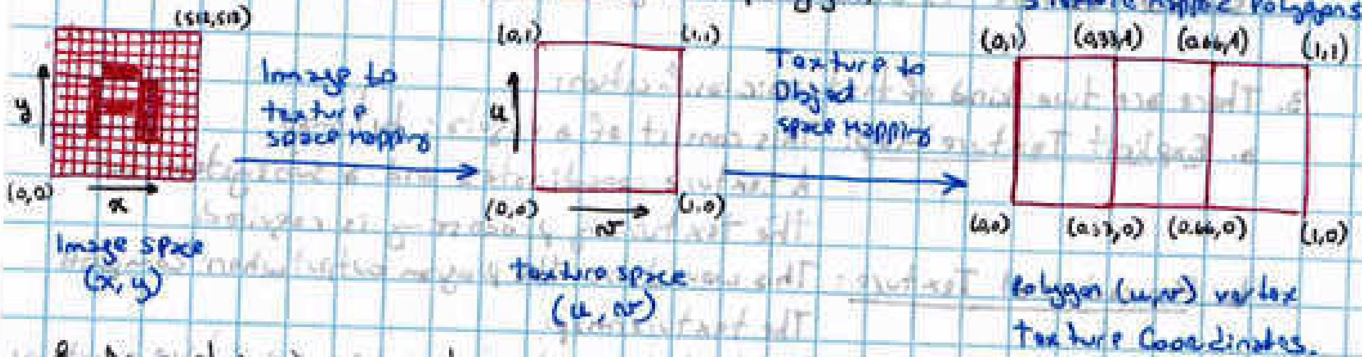
42 7. In a 2D texturing mapping, we map a flat 2D bitmap image onto a surface (flat or curved)

For example:

a. Lets assume we have a 512 x 512 pixels bit map image

b. we wish to assign the texture coordinates (u, v) to their corresponding coordinates onto the diagram containing 3 polygons

c. The 2D Texture mapping process is done by mapping the bitmap pixels to the 3D surface of the polygons.



8. As explained previously we need to map or apply a function that warp an input image into a warped output image.
This means that we begin by measuring the input image in coordinates (u, v) and the resulting warped image in coordinate (x, y)
We must send each coordinate pair (u, v) into the corresponding pair (x, y)
We can express such mapping with two functions:

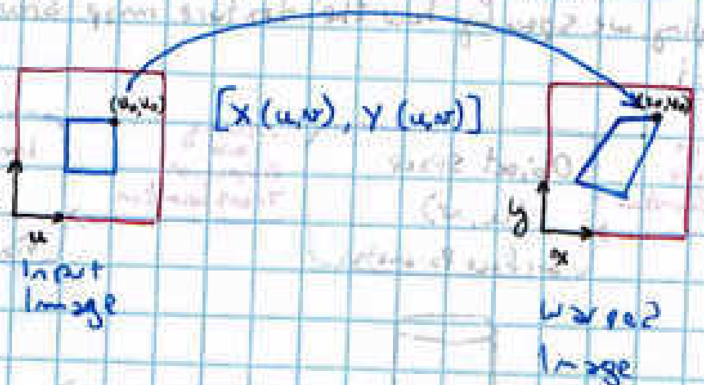
a. function X determines the transformed x coordinate from (u, v)

b. function Y determines the transformed y coordinate from (u, v)

$$x = X(u, v) \qquad y = Y(u, v)$$

or (in vector notation): $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} X(u,v) \\ Y(u,v) \end{bmatrix}$

c. The following is a trivial example of Image map defined by the two previous functions X( ) and Y( ):



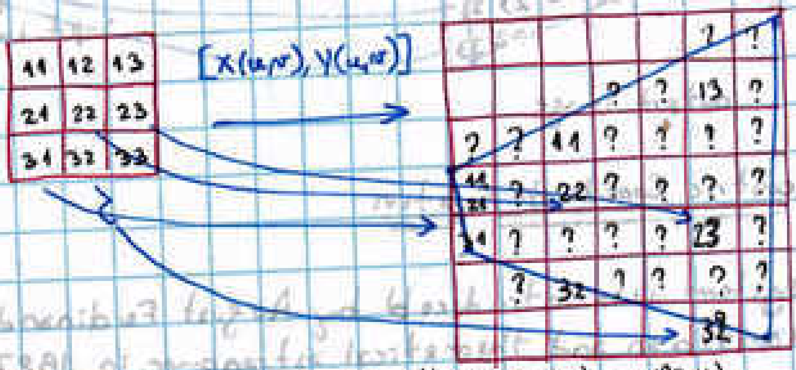
Assignment 4 (continued)

a. We have two ways of mapping:

a. <u>Forward Map</u>: We use each coordinate pair (u, v) to be the point $[X(u,v), Y(u,v)]$ in the output image.

<u>For example</u>:
```
for (v = 0; v < bitmap.height; ++v)
    for (u = 0; u < bitmap.width; ++u)
        output_image [X(u,v)][Y(u,v)] = input_image[u][v];
```

The problem with this approach is that if the image is streched it can leave pixels with unknown values called "holes" as show in the following image:



We must round the coordinates ~~input not the output~~ the given output pixel.

The advantage is that it is easy to implement.

b. <u>Inverse Map</u>: This approach solve the "holes" problems of forward map by ~~sending~~ determining what pixel from the input we have to map first; instead of sending each input pixel(s) map to the output.

Therefore, we must invert the mapping functions $X()$ and $Y()$. The inverse function shall be $U(x,y)$ and $V(x,y)$ such that:

$$u = U(x,y) \qquad \text{or (in vector notation)} \qquad \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} U(x,y) \\ V(x,y) \end{bmatrix}$$
$$v = V(x,y)$$

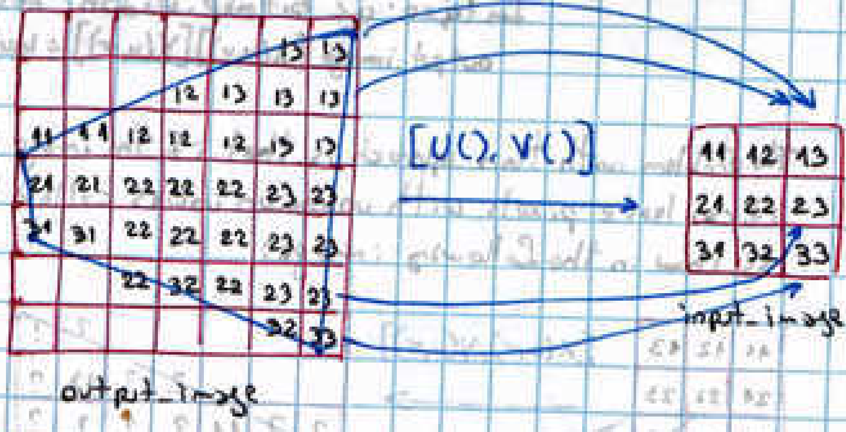
and
$$u = U[X(u,v), Y(u,v)]$$
$$v = V[X(u,v), Y(u,v)]$$
so U and V can be choosen.

At different of forward mapping not only do round the given integer pixel coordinates, but we round the coordinate into the input image and not the output image.

For example:

```
for(y = 0; y < bitmap_output.height; ++y)
  for(x = 0; x < bitmap_output.width; ++x)
    output_image[x][y] = input_image[round(U(x,y))][round(V(x,y))]
```

there is the following figure example:



Barycentric Coordinate System

1. This system was introduced by August Ferdinand Möbius, a german mathematician and theoretical astronomer, in 1827

2. This system helps to locate the barycenter (center point), of a polygon such as a triangle or tetrahedron.

3. Barycentric coordinates are a form of homogeneous coordinates, a system of coordinates used in projective geometry ~~such as~~

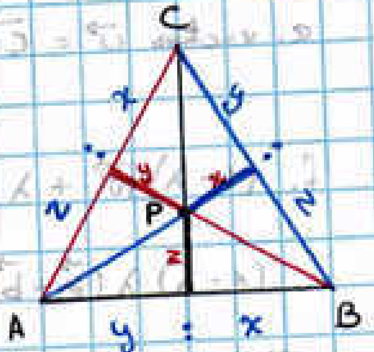
4. In order to explain we describe points in 2D triangle geometry by three coordinates. such as (x, y, z)
Since we are using three numbers instead of two, we add a condition to these coordinates: If $x + y + z = 1$, Then the coordinates are said to be normalized which means that all independent variables are first re-scaled (normalized) so they are inside the range between 0.0 and 1.0
For example: we wish to normalized pixel coordinates

$$\left.\begin{aligned} x_{norm} &= \frac{x-1}{nx-1} \\ y_{norm} &= \frac{y-1}{ny-1} \end{aligned}\right\}$$

were x and y are original pixel coordinates (starting from 0) and nx and ny define the size of the image

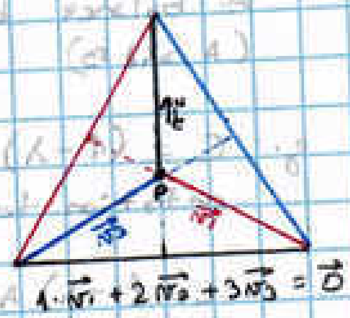
Assignment 4 (continued)

5. Only ratios are used such that $(x,y,z) = (kz, ky, kz)$ where $k>0$.
   We express the coordinate in ratios as follow: $(x:y:z)$



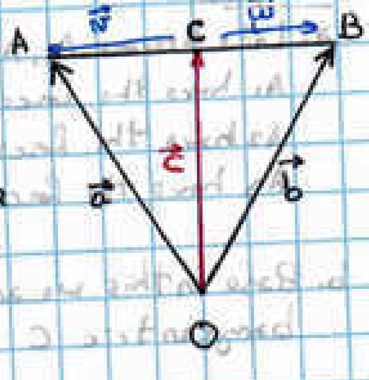
6. In this case the barycentric coordinate $(x:y:z)$ ratio of point P which is the center of the triangle.

7. If the corners A, B, and C are the vectors representing the vertices of the triangle, to write P, write the barycenter coordinate as: $P = Ax + By + Cz$ where $x+y+z=1$ and $A(1,0,0)$, $B(0,1,0)$, and $C(0,0,1)$.



8. Computations in homogeneous coordinates allow us to multiply coordinates by any factor.
   For example: $(bc : ac : ab) = (\frac{1}{a} : \frac{1}{b} : \frac{1}{c})$
   this works since we divide each coordinate by a factor of abc.

9. Coordinates on the affine space:



   a. If you have two points (A and B) on a line and we have some other ~~point~~ extreanol point (O) then we can describe a position on the line between A and B in term of vectors eminating from O.

   b. Point A is represented by vector $\vec{a}$
      Point B is represented by vector $\vec{b}$
      Point C is represented by vector $\vec{c}$

   c. The vector $\vec{c}$ is: $\vec{c} = (1-\lambda)\vec{a} + \lambda b$
      This value ~~determine~~ in which ~~point~~ place the ~~point~~ C is.

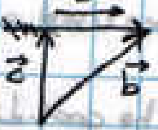
So

d. vector $\vec{v} = \vec{CA} = \vec{a} - \vec{c}$

$= \lambda(\vec{a} - \vec{b})$

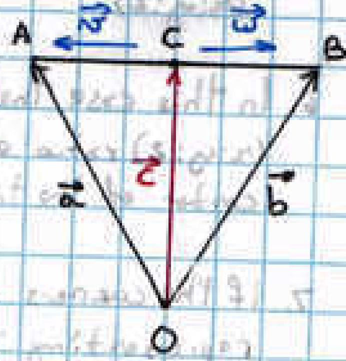
e. vector $\vec{w} = \vec{CB} = \vec{b} - \vec{c}$

$= (1-\lambda)(\vec{b} - \vec{a})$



f. $(1-\lambda)\vec{v} + \lambda\vec{w} = \vec{0}$

$(1-\lambda)\lambda(\vec{a}-\vec{b}) + \lambda(1-\lambda)(\vec{b}-\vec{a}) = \vec{0}$



This tell us that C is the center point of "balance" between 1-λ and λ (A and B)

g. C = (1-λ)A + λB is an alternative point notation to represent $\vec{c} = (1-\lambda)\vec{a} + \lambda\vec{b}$

C = (1-λ)A + λB using points A, B, and C

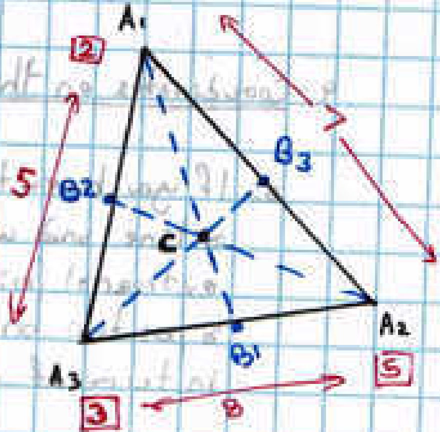
$\vec{c} = (1-\lambda)\vec{a} + \lambda\vec{b}$ using vectors in respect of point O

~~[illegible]~~

10a. Affine with three co-planar triangle.

Example:

a. Lets Assume A_1, A_2, and A_3 are forces such as
A_1 have the force of 2
A_2 have the force of 5, and
A_3 have the force of 3.



b. Base on this we are going to find the barycentric C

c. find middle points on the border of the triangle between points:

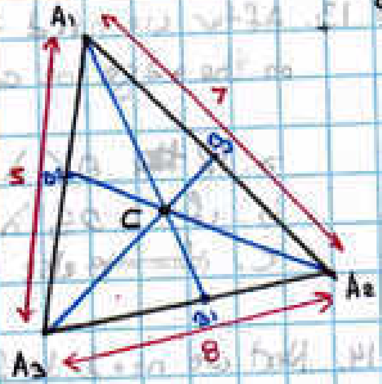
d. $B_1 = \cancel{\frac{3}{5}} \frac{5}{8} A_2 + \frac{3}{8} A_3$

$B_2 = \frac{3}{5} A_3 + \frac{2}{5} A_1$

$B_3 = \frac{2}{7} A_1 + \frac{5}{7} A_2$

$M = 2 + 5 + 10$

Assignment 4 (continued)



10. (continued)

e. $C = \frac{2}{10} A_1 + \frac{5}{10} A_2 + \frac{3}{10} A_3$

$= \frac{2}{10} A_1 + \frac{8}{10} \left(\frac{5}{8} A_2 + \frac{3}{8} A_3 \right)$

$= \frac{2}{10} A_1 + \frac{8}{10} B_1$

$= \frac{5}{10} A_2 + \frac{5}{10} B_2$

$= \frac{3}{10} A_3 + \frac{7}{10} B_3$

11. Special case when $\lambda_1 = \lambda_2 = \lambda_3 = 1/3$

example:

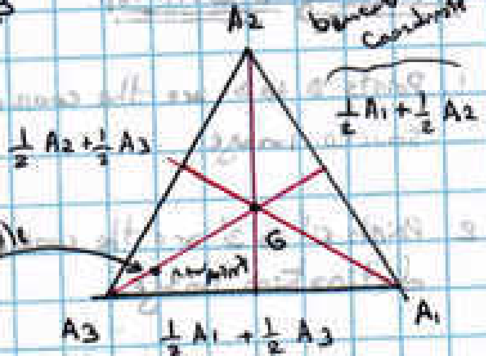
a. $\lambda_1 = \lambda_2 = \lambda_3 = 1/3$

b. $G = \frac{1}{3} A_1 + \frac{1}{3} A_2 + \frac{1}{3} A_3$

$= \frac{1}{3} A_1 + \frac{2}{3} \left(\frac{1}{2} A_2 + \frac{1}{2} A_3 \right)$

$= \frac{5}{6} A_3 + \frac{1}{6} \left(\frac{1}{2} A_1 + \frac{1}{2} A_2 \right)$

example



c. $G = \frac{1}{12} A_1 + \frac{1}{12} A_2 + \frac{5}{6} A_3$

12. For our application, the first step is to calculate λ_1, λ_2, and λ_3 we find them by following these formulas

$$\lambda_1 = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)}$$

$$\lambda_2 = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{(y_3 - y_1)(x_2 - x_3) + (x_1 - x_3)(y_2 - y_3)}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

We keep rotating between the four triangles that are being use in our application

13. After we need to check if the points are inside and/or on the edge or corner of the triangle

   a. If $0 < \lambda_i < 1$ $\forall i$ in 1,2,3 than the point is inside
   b. if $0 \le \lambda_i \le 1$ $\forall i$ in 1,2,3 than the point is on the edge or corner
   c. ~~if~~ else the points reside outside the triangle.

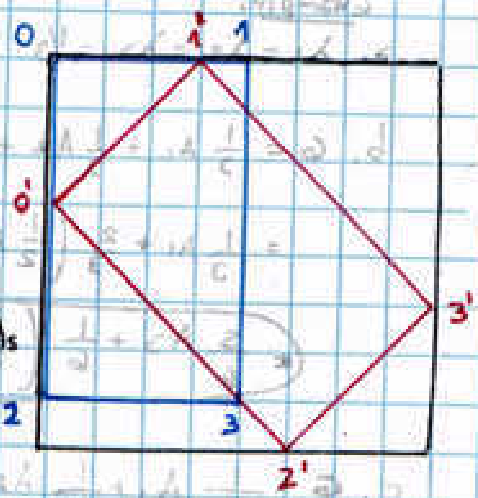
14. Next we need to find the next point base on the new point

$$\left.\begin{aligned} x &= x_1\lambda_1 + x_2\lambda_2 + x_3\lambda_3 \\ y &= y_1\lambda_1 + x_2\lambda_2 + x_3\lambda_3 \end{aligned}\right\} \text{where } x_3 \text{ is the new point } q$$

15. Finally we need to change the offset so it matches the array of bytes that holds the input image used as source.

Rotating a Bitmap

1. Points 0 to 3 are the coordinates of the source image



2. Points 0' to 3' are the coordinates of the destination image.

3. As you may noticed, we must resize the bitmap limits so it can hold the image rotated.

4. The rotation around a pivot could provide us with an unwanted result since we still require to move the bitmap to the center otherwise something more is then required.

5. All angles in degrees must turn into radians: degrees $\cdot \frac{2\pi}{360}$ => radians

6. For this assignment, we are required to rotate the bitmap 45° and then display the texture-mapped (1 image rotated)

7. As extra points we should be able to rotate 360 degrees around z and by (in intervals of 30°)

8. Any time we modify the Bitmap using the pixels, we modify also the size of the new bitmap. So performing 45° one after another increases the image size

Assignment 4 (continued)

9. As explained in point 8, when we rotate 45° degrees the whole image we finish with a new image that is bigger than the original. If we rotate this new image 45 degrees again we finish with the next image that is even more bigger than the original. This is due the "holes" produced by the rotation and resize of the canvas.

Therefore, if we wish to rotate the canvas, we must store the image temporarly and used it over and over again for 45°, 90°, 135°, 180°, 225°, 360°/0°.

10. To obtain the width and height of the destination bitmap, we use the following formulas:

$$new.x = x\cos(\alpha) + y\sin(\alpha)$$
$$new.y = -x\sin(\alpha) + y\cos(\alpha)$$

where the angle is in degrees.

11. In order to implement the cosine and since in programming languages such as C/C++, we are required to convert degrees in radians so: $\text{radians} = \text{degree} \cdot \left(\frac{2\pi}{360°}\right)$

~~12.~~

Assignment 5: Part 1

1. By using the <u>curve interpolation method</u>, we use a <u>cubic parametric polynomial f</u>
$f(u) = au^3 + bu^2 + cu + d$
to fit four points P0, P1, P2 and P3.

This cubic polynomial curve passes through the four control points P0, P1, P2, and P3 when parameter
$u = 0, 1/3, 2/3,$ and 1 respectively.

a. Find a matrix M which satisfies the condition $f(u) = UMP$
~~(where U = [u³, u², u,~~
where • $U = [u^3 \ u^2 \ u \ 1]$
• $P = [P_0 \ P_1 \ P_2 \ P_3]^T$
• $f(u)$ is a vector which has three components $(x(u), y(u), z(u))$
• Each control point P_i ($i = 0, 1, 2, 3$) is a vector representing x, y, z coordinates as (P_{ix}, P_{iy}, P_{iz})

$$f(u) = UMP$$
$$= [u^3 \ u^2 \ u^1 \ 1] \cdot M \cdot [P_0 \ P_1 \ P_2 \ P_3]^T$$

$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$
$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$
$z(u) = a_z u^3 + b_z u^2 + c_z u + d_z$
where $(0 \le u \le 1)$

1. Number of control points $= n+1 = 6$ $\Rightarrow$ $n+1=6$ $\therefore$ $\boxed{n=5}$

2. Degree of polynomial $= d-1 = 2$ (second degree) $\Rightarrow$ $d-1=2$ $\therefore$ $\boxed{d=3}$

3. $P(u) = \sum_{k=0}^{n} P_k B_{k,d}(u)$ (where: $d \in [2, n+1]$
$k = \{0, \dots, n\}$

4. $\begin{cases} B_{k,1}(u) = 1, & \text{if } u_k \le u < u_{k+1} \\ \phantom{B_{k,1}(u)} = 0, & \text{otherwise} \end{cases}$

5. $$B_{k,d}(u) = \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u)$$

(where $u_k \le u < u_{k+d}$

5. Continuity $= C^{d-2} = C^{3-2} = C$

6. Blending functions ~~are quadratic $P(0) \to P(n-1)$~~ B
~~$\{P_0, P_1, \dots, P_4\}$~~
↳ ~~$P(0), P(1), P(2), P(3), P(4)$~~

~~7. Number of knot values: $n+d+1 = 5+3+1 = 9$ $\therefore$ $\{u_0, u_1, \dots, u_9\}$~~

7. Number of knot values $= n+d+1 = 5+3+1 = 8+1 = 9$

Number of divisions of subintervals $= n+d = 5+3 = 8$

$\{u_0, u_1, \dots, u_{n+d}\}$
$\Downarrow$
$\{u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$

- "Uniform" distribution of knots are done by spacing them in equal intervals of the parameter u

54

$n=5$
$d=3$

$$\left[\text{If } B_{k,d}(u) = \frac{u-u_k}{u_{k+d-1}-u_k} \cdot B_{k,d-1} + \frac{u_{k+d}-u}{u_{k+d}-u_{k+1}} B_{k+1,d-1}(u) \quad (\text{where } u_k \le u < u_{k+d}) \right]$$

$$B_{0,3} = \frac{u-u_0}{u_2-u_0} B_{0,2}(u) + \frac{u_3-u}{u_3-u_1} B_{1,2}(u)$$

$$B_{1,3} = \frac{u-u_1}{u_3-u_1} B_{1,2}(u) + \frac{u_4-u}{u_4-u_1} B_{2,2}(u)$$

$$B_{2,3} = \frac{u-u_2}{u_4-u_2} B_{2,2}(u) + \frac{u_5-u}{u_5-u_2} B_{3,2}(u)$$

$$B_{3,3} = \frac{u-u_3}{u_5-u_3} B_{3,2}(u) + \frac{u_6-u}{u_6-u_3} B_{4,2}(u)$$

$$B_{4,3} = \frac{u-u_4}{u_6-u_3} B_{4,2}(u) + \frac{u_7-u}{u_7-u_4} B_{5,2}(u)$$

$$B_{5,3} = \frac{u-u_5}{u_7-u_3} B_{5,2}(u) + \frac{u_8-u}{u_8-u_5} B_{6,2}(u)$$

$$u_i = \begin{cases} 0, & \text{if } i<d \\ i-d+1, & \text{if } d \le i \le n \\ n-d+2, & \text{if } i>n \end{cases}$$

$\boxed{u_0} = 0 \quad ,(i<d)$

$\boxed{u_1} = 0 \quad ,(i<d)$

$\boxed{u_2} = 0 \quad ,(i<d)$

$\boxed{u_3} = 3-3+1 = \boxed{1}, (d \le i \le n)$

$\boxed{u_4} = 4-3+1 = \boxed{2}, (d \le i \le n)$

$\boxed{u_5} = 5-3+1 = \boxed{3}, (d \le i \le n)$

$\boxed{u_6} = 5-3+2 = \boxed{4}, (i>n)$

$\boxed{u_7} = 5-3+2 = \boxed{4}, (i>n)$

$\boxed{u_8} = 5-3+2 = \boxed{4}, (i>n)$

$u_i: \{0,0,0,1,2,3,4,4,4\}$

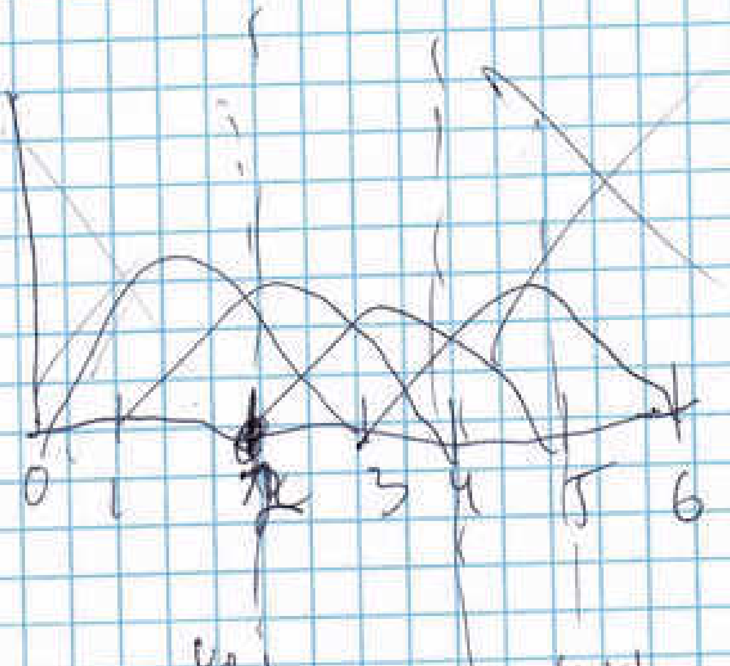
$$\left[\cancel{B_{6,3} = \frac{u-u_6}{u_8-u_3} B_{6,2}(u) + \frac{u_9-u}{u_9-u_5} B_{7,2}(u)} \right]$$ doesn't exist

---

$$B_{0,2} = \frac{u-u_0}{u_1-u_0} B_{0,1} + \frac{u_2-u}{u_2-u_1} B_{1,1}$$

$$B_{0,1} = \frac{u-u_0}{u_0-u_0} B_{0,0} + \frac{u_1-u}{u_1-u_1} B_{1,0}(u) = 0$$

$$B_{1,1} = \frac{u-u_1}{u_1-u_1} B_{1,0} + \frac{u_2-u}{u_2-u_2} B_{2,0}(u) = 0$$



$a0 + b0 + c0 + d$

$a(0)^3 + b(0)^2 + c(0) + d$

$f(u) = au^3 + bu^2 + cu + d$

2

6

4

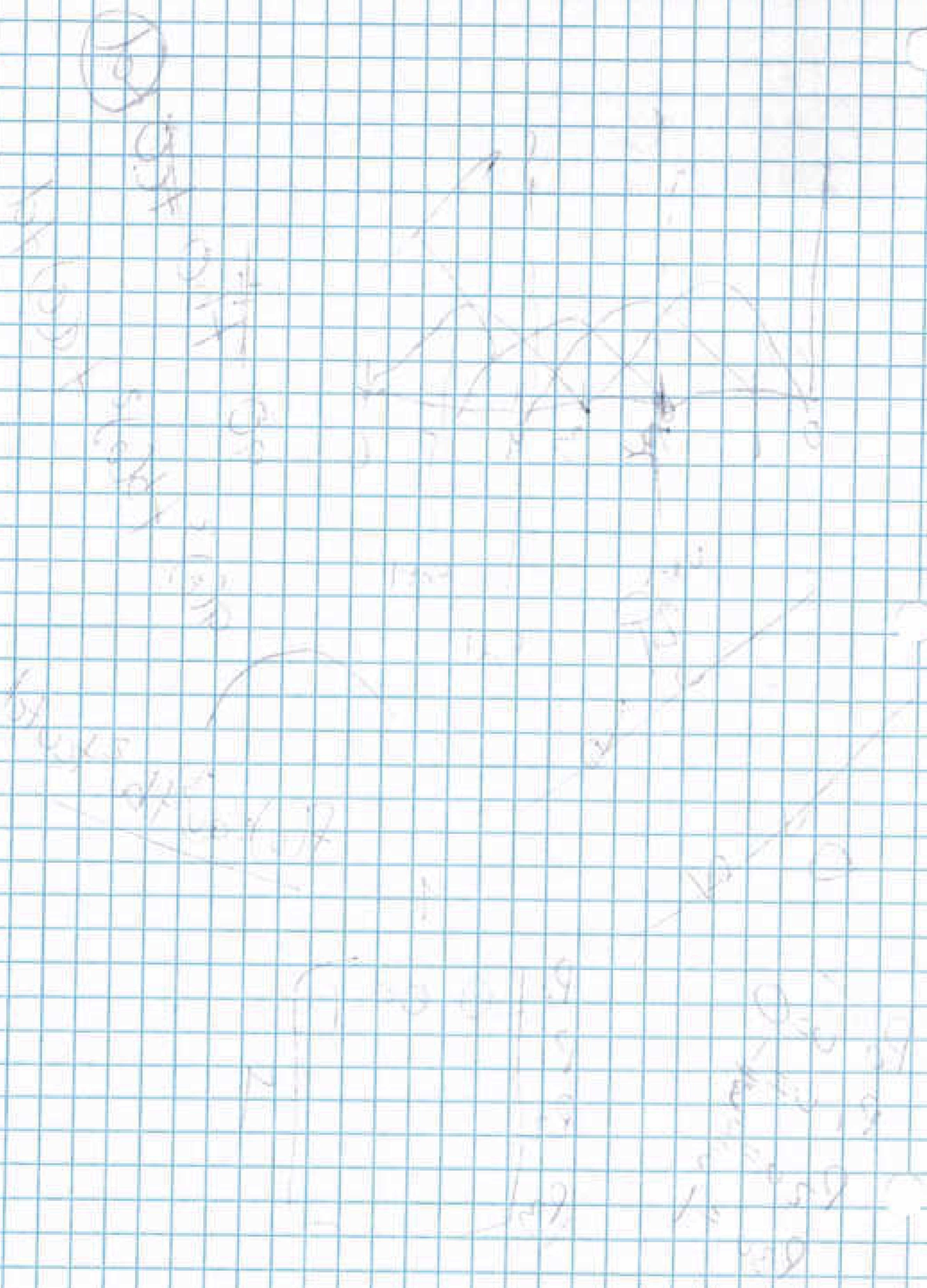
$P_0 \; u = 0$

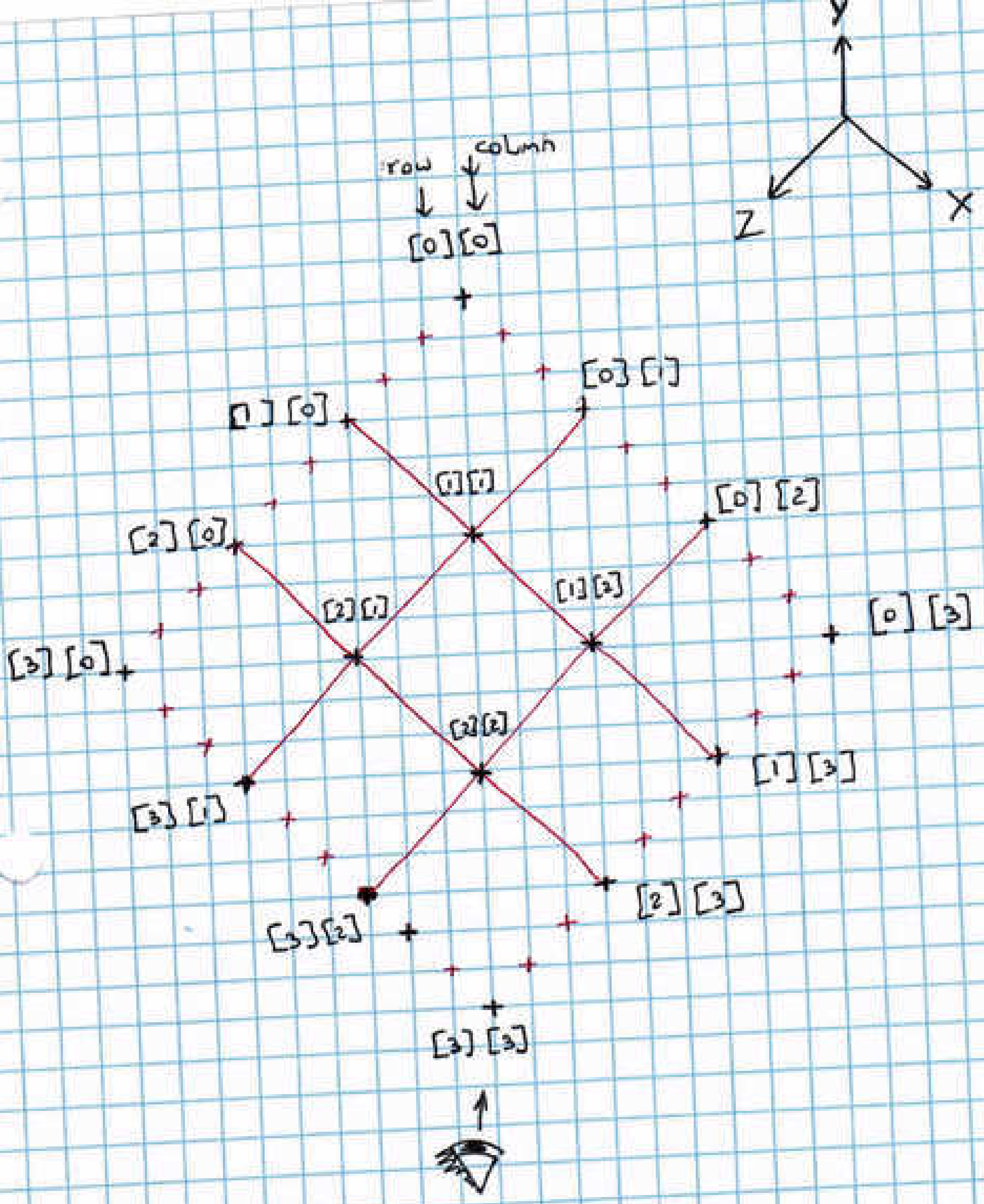
$P_1 \; u = \frac{1}{3}$

$P_2 \; u = \frac{2}{3}$

$P_3 \; u = 1$

$$\begin{array}{c} P_0 \\ P_1 \\ P_2 \\ P_3 \end{array} \begin{bmatrix} 0 & 0 & 0 & 1 \\ & & & \\ & & & \\ & & & \end{bmatrix} 4$$




y
X
Z
row
colmn
[0] [0]
[0] [1]
[1] [0]
[1] [1]
[0] [2]
[2] [0]
[1] [2]
[2] [1]
[0] [3]
[3] [0]
[2] [2]
[1] [3]
[3] [1]
[2] [3]
[3] [2]
[3] [3]