Laboratório VISGRAF
Instituto de Matemática Pura e Aplicada

Stitching and Visualization of 360 Images

Aldo Nogueira
Luiz Velho (supervisor)

Technical Report TR-02-03 Relatório Técnico

March - 2002 - Março

Stitching and Visualization of 360° Images at Fotorama®

Aldo Nogueira

May 14, 2002

This document describes the algorithms used in the Fotorama software to make 360° panoramic images.

1 Introduction

Imagine you own a hotel and want to show how it is to possible customers worldwide. Putting single photos of the entrance hall and rooms in the web site does not give a good idea of these places. A better way to give a good feeling of a space is to use panoramas. With one of these and a appropriate viewer, you can look to any direction you want from one point of view. Getting one from the entrance of your hotel, you can show anyone who has a browser, how beautiful is its front, the street, the gardens, the sky, etc. Everything around the tripod is captured.

The process begins when we take two photos of a place. The hardware is a digital camera with fisheye lenses, a tripod and a special support. This lenses have a 183° field of view. This means that with one photo, you get a bit more than a half of the whole environment in a image. Rotate the camera of 180° horizontally (the special support helps you here) and take a second photo. Now you have information from any direction around a point. The support and tripod guarantees that rotation is done around lenses nodal point.

The second phase is to join these two images in a process called stitching. That extra 3° in each photo is very important now. Here we calculate how the two images fit with each other and adjust colors to generate only one image containing everything. This resulting image is distorted, a little difficult to understand.

Finally we have to view it. The viewer program do some undistortions on the result image to simulate a virtual camera on the original environment. The user can move the mouse to control the camera and look anywhere he/she wants.

2 Stitching

The process of joining the two fisheye photos in a 360° panorama is called stitching. This is done with a program written in C++ to get best performance in both CPU and memory management.

The first step is cropping the original photos to get only the part that contains useful information. The whole environment captured by the shot is inside a circle in the photos. We have to find this circle and crop the photos in a square around them.

The second step is to warp the cropped images. By doing it we are using a coordinate system that transform images in such way that those redundant 3° around the images are put in similar form in plane. This is important because it is easier to compare these images in the next step.

The third step is to find how to fit the two images to compose the panorama taking care of adjusting colors in both images to make the junction invisible. We already can use this panorama in a viewer.

Sometimes is interesting to apply a forth step to make the panorama easier to understand to humans. We just rotate the image in 3d space to get the ground below and sky on top.

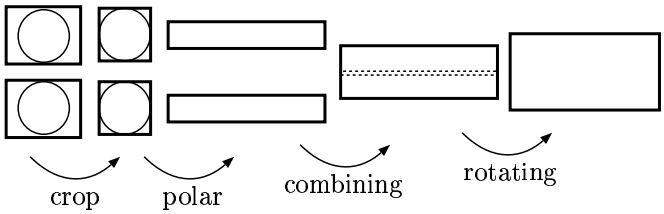

Figure 1: The whole process.

2.1 Cropping

The first step applied to the two images is to extract the part relevant to stitching. They come as shown in figure 2. The two images are very similar. The whole information from the environment is on a circle. The area around it is not exposed to light and them it is entirely black. Our goal here is to find where this circle is. Its location depends on what camera you are using, what lenses and may also vary because little mechanical adjustments. So it may be different between different pairs of photos but we expect they are quite the same within a pair. So we try to find the circle using a combined image C which pixels are obtained from the original images I_1, I_2 as in (1).

$$C(i,j) = \frac{I_1(i,j) + I_2(i,j)}{2} \tag{1}$$

A way we could use to find this circle is walk through the image from borders to the center searching for a pixel brighter than black. Repeating this three times

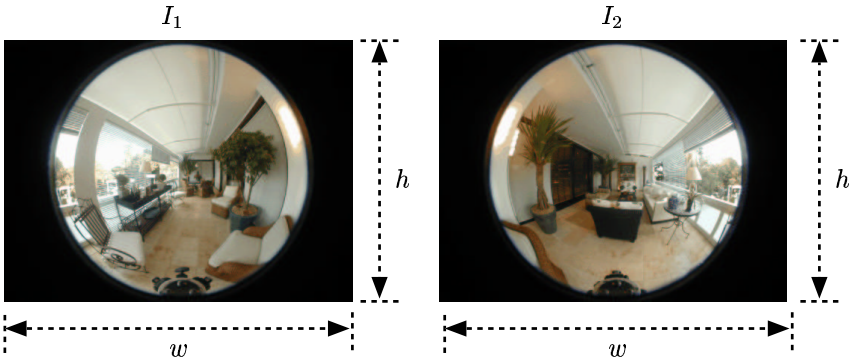

Figure 2: The images how we get them from the camera.

through different directions, we should get three points on the edge of the circle. We can use them to find circle dimensions. Actually we search for the first pixel which pseudo-gradient g is greater than a threshold. We will talk about this later.

How we get circle dimensions from three points on its edge: We could name them P_1, P_2 and P_3. Now let m_1 and m_2 be the medians of $\overline{P_1P_2}$ and $\overline{P_2P_3}$ segments respectively. See figure 3. The center lies on the intersection of m_1 and m_2. The radius is the distance between C and one of the points P_1, P_2, P_3.

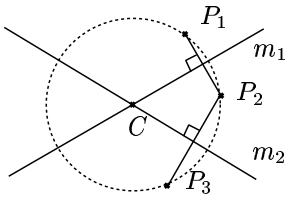

Figure 3: Getting circle center from three points.

Just three points isn't enough to find the circle because the black area sometimes is not entirely black or inside the circle exist really dark regions (some black object e.g). See figure 4. To find the circle in spite of previous problems, we need to obtain many points and select triples from them and make something like to find the average circle.

Discarding every circle with absurd dimensions (too small, too big, center too distant from image center), we collect n of them and make an average circle. See (2).

$$\begin{cases} x = \sum_{i=1}^{n} x_i/n \\ y = \sum_{i=1}^{n} y_i/n \\ r = \sum_{i=1}^{n} r_i/n \end{cases} \tag{2}$$

Figure 4: Light and dark regions where they should not be.

The pseudo-gradient g is got from following expression:

$$g(x,y) = \sum_i \sum_j I(x+i, y+j) \times \phi(i,j) \times \begin{pmatrix} i \\ j \end{pmatrix} \tag{3}$$

where I is the image and

$$\begin{cases} \phi(i,j) = \cos(d \times \pi/2)/(1+d) & \text{if } d < 1 \\ \phi(i,j) = 0 & \text{if } d \geq 1 \end{cases}$$

where

$$d = \sqrt{p^2 + q^2}/r$$

and r is the radius of where to analise image. We did find good results with $r = 2$.

2.2 Polar Warping

Now the idea is to apply polar warping to cropped images. We will transform polar coordinates to rectangular ones. The result image have $2\pi r$ of width and r of height, adopting r as the circle radius. Remember that crop dimensions are $2r \times 2r$. The destination image pixels are found from the source image by the expression in (4).

$$I'(i,j) = I(j \times \sin(i/r - \pi) + r, j \times \cos(i/r - \pi) + r) \tag{4}$$

The first image circle border is mapped on the bottom and center on top. The second one is the same thing but after polar filter, we rotate it in 180°. See figure 5. This important to the next steps of this algorithm.

We use bi-linear interpolation in this operation.

2.3 Fitting

A portion of the photographed environment is registered repeated on two photos. This is the extra 3° on them. Then the area around the circle border in both

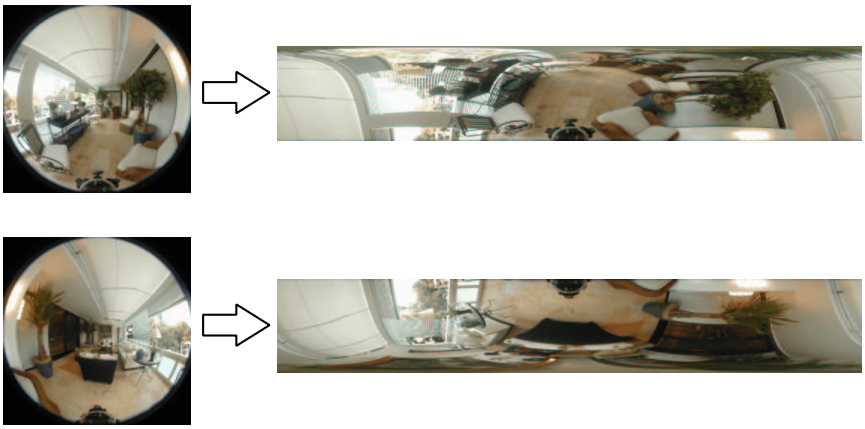

Figure 5: Polar warping.

photos are equivalent. Apply the polar filter as described earlier, we obtain two images that fit with each other. The bottom of the first one fits on top of the second one. Now we need to know how much we have of intersection. The algorithm was only this in earlier versions. We will discuss this problem now and we will deal with other ones later.

Let's define a function $e(y)$ that determines the error of images' junction from the images I_1 e I_2 and from how much they intercept.

$$e(y) = \sum_{i=0}^{w-1} \sum_{j=0}^{y-1} \frac{|I_1(i, h_1 - j - 1) - I_2(i, j)|^2}{w \times y} \tag{5}$$

Where w is the width of the images, y is the intersection in pixels and h_1 is the height of the first image. The distance in (5) is the euclidean distance between colors (points in $\mathbb{R}^3$) of images at that pixels.

The idea is just to vary y from a minimum value to a maximum one and calculating the error. We adopt the intersection between images as that one that gives smallest error.

There are some photos that do not fit using this algorithm. Thus, we introduce a horizontal displacement x. The error function now becomes like in (6). Now we also vary x within a range that we think it is reasonable.

$$e(x, y) = \sum_{i=0}^{w-1} \sum_{j=0}^{y-1} \frac{|I_1(i + x, h_1 - j - 1) - I_2(i, j)|^2}{w \times y} \tag{6}$$

When $i + x$ is greater than the image width, we subtract w to connect the right side with the left one. It is like rotating the camera around viewing axis.

Because of imprecisions of the camera's tripod and handling problems, the two photos are not rotated exactly by 180°. Some do not fit because of it.

For this reason, we introduce a 3d rotation (See figure 6) to compensate this misalignment.

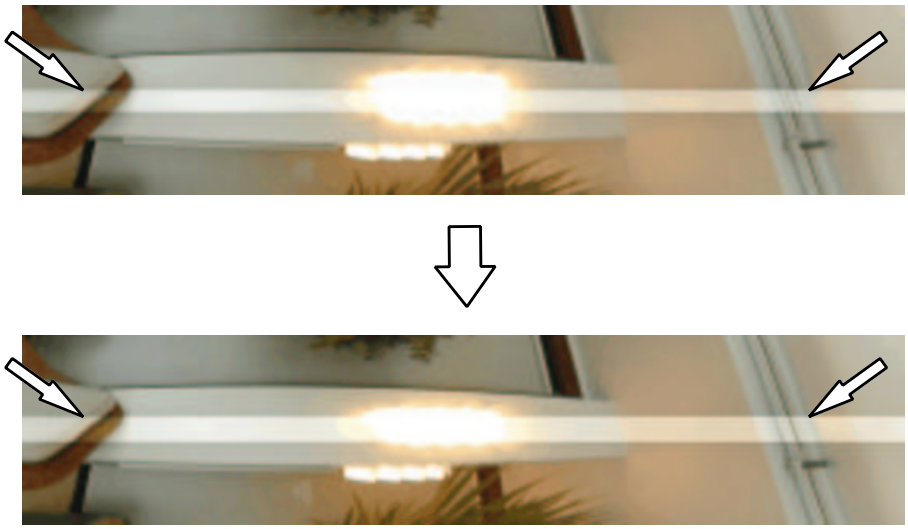

Figure 6: The realignment (the lighter region is the intersection).

$$I' \begin{pmatrix} i \\ j \end{pmatrix} = I \left(t^{-1} \left(M \times t \begin{pmatrix} i \\ j \end{pmatrix} \right) \right) \tag{7}$$

Where t is the transformation that maps the point in image coordinates to a point in a sphere of radius 1 and center at the origin. See (9) e (10). $M = \text{rot}_y(a) \times \text{rot}_x(b) \times \text{rot}_y(-a)$. Notice that $\text{rot}_y(a)$ is a rotation matrix of a radians around y axis.

The algorithm now has four degrees of freedom: the earlier x and y and the two angles a and b. First we rotate the second image according a and b and then we apply the process described above. One for each pair a and b. We note the values that give the smallest error, rotate the second image according and pass to the next step.

2.4 Combining and Adjusting Colors

The input of this algorithm is the two realigned images and two integers: the intersection n and the horizontal displacement m. We want to generate only one image from these data.

The top portion of the resulting image I_r comes from the first source image and the bottom portion comes from the second one. The middle portion, that is a band with n pixels of width, is a blending from one image to the other one. The second image is displaced in m pixels horizontally before we start the process. The height of the final image is $h_1 + h_2 - n$.

The pixels of the middle portion are defined as:

$$I_r(i,j) = \begin{cases} I_1(i,j) & \text{if } j < h_1 - n \\ I_1(i,j) \times f + I_2(i, j - h_1 - n) \times (1 - f) & \text{if } h_1 - n < j < h_1 \\ I_2(i, j - h_1 - n) & \text{if } j > h_1 \end{cases} \tag{8}$$

Sometimes the images do not have similar colors and then the junction becomes too visible. This is why we need to compare the colors in the intersection region of the two images and to make them closer to each other.

3 Visualization

A 360° panorama is not easy to interpret visually, so we would like to use a special viewer. Our viewer was made in Java to be small and portable. It runs on Linux, MacOS, AIX and Windowz with Mozilla, Konqueror, Netscape, IE, etc.

The viewer simulates a virtual camera inside a sphere or a cube mapped with the panorama. When the user moves the mouse, the viewer changes virtual camera position and take another shot. This is done fast enough to achieve interactive frame rates in any machine equal or faster than a AMD K6.

3.1 Spheric Projection

Here, first we will desbribe how we map a panorama image to an sphere and after, we will be concerned about the virtual camera inside that sphere.

We map the image to the sphere, just like a world map is mapped into a globe. See Figure 7. In fact, the only difference is that the interesting side on a globe is the outer one and here we are interested with the inner one and thus, longitude is inverted.

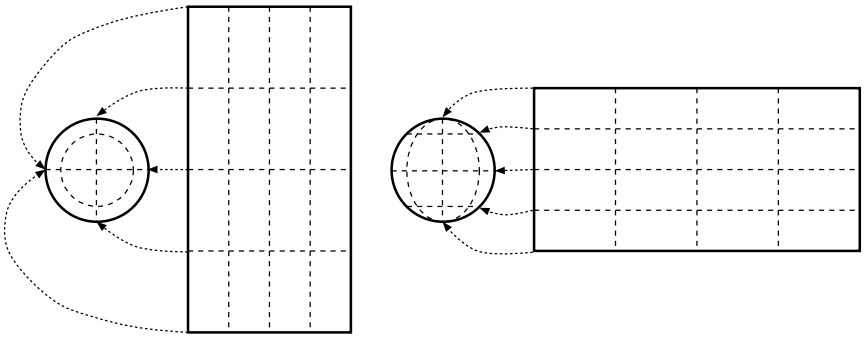

Figure 7: Mapping the panorama image to the sphere.

We map (x_i, y_i) from the image's coordinate system to (θ, ϕ) (theta and

phi) that would respectively represent longitude and latitude in radians on that globe. See Figures 8 and 9.

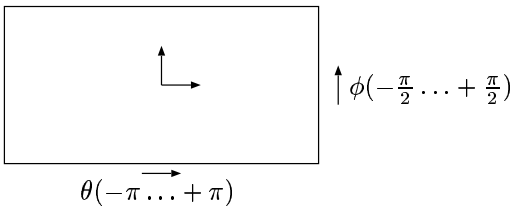

Figure 8: Mapping (θ, ϕ) to the image.

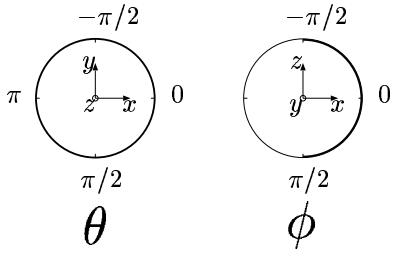

Figure 9: Mapping (θ, ϕ) to the sphere.

When we have a sphere with that panorama image mapped to its inner surface, we can just put inside a little camera which focus is on the sphere centre. Any photos made with this camera would be like it were on the place where the original photos were taken.

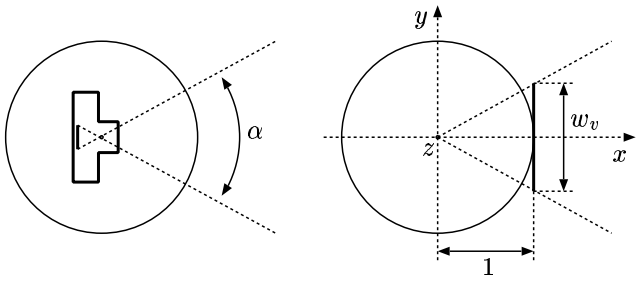

Figure 10: The camera inside the sphere.

This camera is described by 6 parameters:

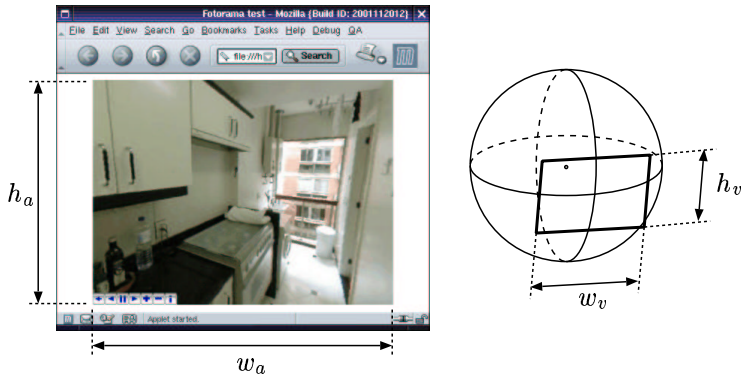

Figure 11: The virtual screen placed on sphere.

$$\left\{\begin{array}{ll} \left.\begin{array}{l}\theta_0\\ \phi_0\end{array}\right\} & \text{define what direction it is pointed like on Figure 12}\\ \alpha & \text{camera aperture like on Figure 10}\\ zoom & \\ \left.\begin{array}{l}w_a\\ h_a\end{array}\right\} & \text{width and height in pixels of the image to be generated}\end{array}\right.$$

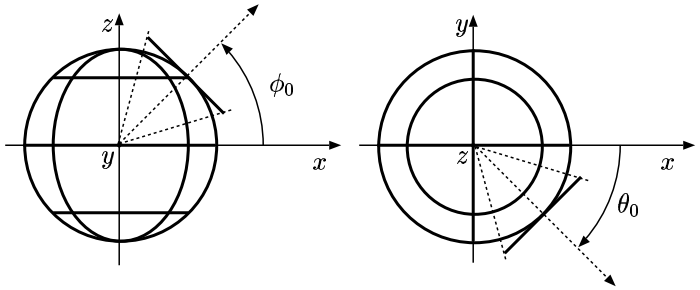

Figure 12: How ϕ_0 and θ_0 affects the virtual screen position.

The equations (9) and (10) represent the conversion between the map coordinate system and 3D cartesian coordinate system on a sphere that would be described by $x^2 + y^2 + z^2 = 1$.

$\underline{(\theta, \phi) \rightarrow (x, y, z)}$

$$\left\{\begin{array}{l} x = \sin(\theta) \times \cos(\phi)\\ y = \cos(\theta) \times \cos(\phi)\\ z = \sin(\phi)\end{array}\right. \tag{9}$$

$\underline{(x,y,z) \rightarrow (\theta,\phi)}$

$$\begin{cases} \theta = \begin{cases} \pi/2 & \text{if } x = 0 \text{ and } y > 0 \\ -\pi/2 & \text{if } x = 0 \text{ and } y < 0 \\ \arctan(y/x) & \text{if } x > 0 \\ \arctan(y/x) - \pi & \text{if } x < 0 \text{ and } y \leq 0 \\ \arctan(y/x) + \pi & \text{if } x < 0 \text{ and } y > 0 \end{cases} \\ \phi = \arcsin(z) \end{cases} \tag{10}$$

From output image dimensions (w_a and h_a), camera aperture (α) and $zoom$ we can obtain virtual screen dimensions (w_v and h_v). See equation (11) and Figures 10 and 11.

$$\begin{cases} w_v = \frac{2 \times \tan(\alpha/2)}{zoom} \\ h_v = \frac{w_v \times h_a}{w_a} \end{cases} \tag{11}$$

With w, h, ϕ_0, θ_0 (see figure 12) we can map output image pixels (x_a, y_a) to virtual camera (x_v, y_v, z_v). First we map it to a virtual screen with $\phi_0 = 0$ and $\theta_0 = 0$ using equation (12) and (13).

$\underline{(x_a, y_a) \rightarrow (x_v, y_v, z_v)}$

$$\begin{cases} x'_a = (x_a - \frac{w_a}{2}) \times \frac{w_v}{w_a} \\ y'_a = (\frac{h_a}{2} - y_a) \times \frac{h_v}{h_a} \end{cases} \tag{12}$$

$$\begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} = (x'_a \times -\vec{y} + y'_a \times \vec{z}) + \vec{x} = \begin{pmatrix} 1 \\ -x'_a \\ y'_a \end{pmatrix} \tag{13}$$

Now we use ϕ_0, θ_0 to rotate it to its correct position. See equation (14).

$$\begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} = M_2 \times M_1 \times \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \tag{14}$$

Where M_1 and M_2 are rotation matrices around y and z axes respectively. See them following: equations (15) and (16).

$$M_1 = \begin{pmatrix} \cos(\phi) & 0 & -\sin(\phi) \\ 0 & 1 & 0 \\ \sin(\phi) & 0 & \cos(\phi) \end{pmatrix} \tag{15}$$

$$M_2 = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{16}$$

Now we will see the inverse mapping, i. e., from a point (x_v, y_v, z_v) to obtain the corresponding (x_a, y_a) on the applet screen. First we rotate it back to position $\phi_0 = 0$ and $\theta_0 = 0$. See equation (17).

$\underline{(x_v, y_v, z_v) \rightarrow (x_a, y_a)}$

$$\begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} = M_1^{-1} \times M_2^{-1} \times \begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} \tag{17}$$

Before mapping it to applet screen we must project it in $x = 1$. See equations (18), (19) and (20).

$$\begin{pmatrix} 1 \\ -x'_a \\ y'_a \end{pmatrix} = \begin{pmatrix} x_0/x_0 \\ y_0/x_0 \\ z_0/x_0 \end{pmatrix} \tag{18}$$

$$\begin{cases} x'_a = -y_0/x_0 \\ y'_a = z_0/x_0 \end{cases} \tag{19}$$

$$\begin{cases} x_a = x'_a \times W/w + W/2 \\ y_a = -y'_a \times H/h + H/2 \end{cases} \tag{20}$$

Mapping θ and ϕ to the image is like on figure 8. see it on equations (21) and (22), where w_i and h_i stand for the image dimensions.

$(\theta, \phi) \rightarrow (x_i, y_i)$

$$\begin{cases} x_i = \frac{\theta}{\pi} \times w_i/2 + w_i/2 \\ y_i = \frac{2\phi}{\pi} \times h_i/2 + h_i/2 \end{cases} \tag{21}$$

$(x_i, y_i) \rightarrow (\theta, \phi)$

$$\begin{cases} \theta = \pi \times \frac{x_i - w_i/2}{w_i/2} \\ \phi = \pi/2 \times \frac{-y_i - h_i/2}{h_i/2} \end{cases} \tag{22}$$

The whole process is described following.

$$\underbrace{(x_a, y_a)}_{\text{applet screen}} \rightarrow \underbrace{(x_v, y_v, z_v)}_{\text{virtual screen}} \rightarrow \underbrace{(\theta, \phi)}_{\text{sphere}} \rightarrow \underbrace{(x_i, y_i)}_{\text{panorama image}}$$

$(x_a, y_a) \rightarrow (x_v, y_v, z_v)$ Equations (12), (13) and (14).
$(x_v, y_v, z_v) \rightarrow (\theta, \phi)$ Equation (10).
$(\theta, \phi) \rightarrow (x_i, y_i)$ Equation (21).

$$\underbrace{(x_i, y_i)}_{\text{panorama image}} \rightarrow \underbrace{(\theta, \phi)}_{\text{sphere}} \rightarrow \underbrace{(x_v, y_v, z_v)}_{\text{virtual screen}} \rightarrow \underbrace{(x_a, y_a)}_{\text{applet screen}}$$

$(x_i, y_i) \rightarrow (\theta, \phi)$ Equation (22).
$(\theta, \phi) \rightarrow (x_v, y_v, z_v)$ Equation (9).
$(x_v, y_v, z_v) \rightarrow (x_a, y_a)$ Equations (17), (18), (19) and (20).

3.2 Cubic Projection

The cubic projection have two main advantages over the spheric one: memory savings and viewing speed.

On earlier versions of the viewer, the projection was spheric as described on previous section, then the algorithm input was the image as it comes from stitching. The image needs to be processed before it enters the cubic viewer. We make use of the spheric viewing algorithm to take 6 'photos' of the environment: bottom, top, left, right, front and back. These photos are squares and the camera is set to 90° of aperture. They can be mapped on a cube, respecting relative positions, in its faces. A virtual camera with center of projection inside this cube is able to take photos just like it were on the sphere center.

The memory savings occur because on the sphere poles, there are too many pixels in a little viewing area. On the cube, the change in density of pixels per vieweing angle is much smaller. On cube corners, the density is a little higher than in the face center. See figure 13.

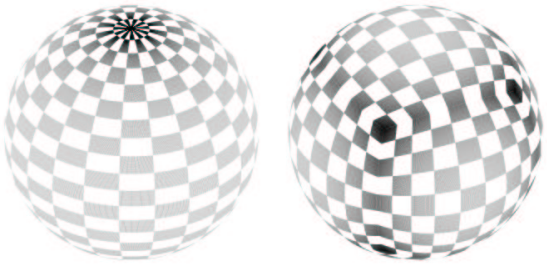

Figure 13: Pixel density on spheric and cubic projection

The speed increase happens because, for each pixel of finel image, on spheric projection we must calculate two arctans and one square root(expensive operations) while on the cubic one, are only some a few sums and products. Moreover, we can use only integer operations in some circunstances.

Figure 14: The mapped cube.

Obs.: This is a separable transformation. We can exploit this property to get better performance.

3.3 The .fot format

The input of our viewer is a file which contains a panorama. In its header are the file size, initial position, camera aperture, maximum and minimum zoom and rotating speed. After these data we have six images that are mapped on the cube faces. At the end are the hotspots, links to other panoramas or any URL.

List of Figures