Author: Ünsal Öztürk

Taken from an internship report.

1.1 Methods for Achieving of Depth of Field Effects

1.1.1 Aperture Sampling for Depth of Field in Off-line Rendering

A frequently used method to obtain depth of field effects in off-line rendering in the context of physically based Monte Carlo light transport simulation is aperture sampling, a detailed explanation of which is given in PBRT [1]. This method works by increasing the size of the aperture of the camera from an infinitesimally small point (aka the pinhole camera model) to a disk of small (comparatively small in comparison to the geometry of the scene) size. Instead of shooting rays into the scene from a point, the starting position of the rays are now determined through the sampling of a particular shape representing the aperture of the camera. This is not enough, however, since as discussed before, the depth of field effect results from the nature of the lens system in the camera, and therefore it is necessary to simulate the lens system as well. A common approach is to mimic the behavior of an infinitesimally thin thin-lens. In this setup, there are two parameters to achieve a physically correct depth of field effect: the size of the aperture and the distance between the aperture and the focal plane of the lens. An example implementation would choose either a hexagon or a disk as the shape of the aperture, it would then sample these shapes uniformly (or may use some clever importance sampling if the scene is known before-hand) by using the inversion method or rejection sampling, and then project the rays on the focal plane of the thin-lens. This results in a very good approximation of the effect as present in real cameras, although effects such as chromatic aberrations or lens distortions are unobtainable with this setup.

1.1.2 Approximation Methods for Real-Time Depth of Field Effects

The framework described in the previous section is unfortunately quite difficult to obtain using real-time rendering pipelines, such as DirectX or OpenGL, and therefore some cheaper approximations are used to create visually convincing -however physically incorrect- depth of field effects. This section discusses two subclasses of these methods.

1.1.2.1 Multiple Renders for a Physically Correct Real-Time Depth of Field Approximation

The motivation for this method was established by Haeberli and Akeley in their accumulation buffer paper [2]. This algorithm is an imitation of aperture sampling as described previously for off-line graphics pipelines. While it is a better approximation in comparison to other methods employed for real time depth of field effects, it is extremely expensive, and it drastically reduces render-times. The algorithm is as follows:

```
get_depth_of_field_image(camera, theta, aperture_size, focal_point) {
      vector4f u = camera.up;
      point4f o = camera.origin;
      matrix4f r = get_rotation_matrix(camera.look_at, camera.origin,
      theta);
      buffer4f accum_img;
      for (int i = 0; i < floor(2*pi/theta); i++) {
            camera.origin = o + aperture_size * u;
            camera.look_at(focal_point);
            accum_img += render_image();
            u = matrix_multiply(r*u).normalized();
      }
      camera.origin = o;
```

```
        camera.look_at(focal_point);
        return accum_img /floor(2*pi/theta);
}
```

What this algorithm does is that it renders the scene multiple times by placing the camera around a circle centered at the origin of the camera and perpendicular to the initial look-at vector of the camera, with a predetermined radius of `aperture_size`, with each camera placement being `theta` radians apart on this circle for each render. The images obtained are accumulated in a buffer and then they are averaged at the end of the algorithm, before resetting the camera parameters to their original values.

This algorithm approximates aperture sampling for small aperture sizes and theta values, and produces a depth of field which is almost physically correct (focal plane centered at the depth coinciding with the middle of the screen space) at the cost of multiple renders. For instance, setting theta to π radians reduces frames rendered per second by half, since the scene is rendered two times. This algorithm is therefore inapplicable in most cases; however, it is possible to obtain some effects such as the partial occlusion effect produced by objects lying on the near plane and circles of confusion are implicitly calculated.

1.1.2.2 Depth Buffer Based Blurring Methods

These methods use depth buffer information to appropriately blur certain parts of the rendered image. The depth of field effects obtained through using these efforts are physically incorrect, however they are visually good approximations and if implemented correctly, they are incredibly cheap and work well as post-processing effects. A compilation of these methods is given in [3].

Common pipelines for these methods include, but are not limited to:

- Separation of the image into focal, near, and far planes: The depth buffer is used to determine which fragments/pixels are beyond the focal plane, around the focal plane, and in between the focal plane and the camera. These planes are usually blurred using filters of different sizes, and then are blended by rendering these planes in a specific order and using alpha blending to reduce extremely sharp transitions.

- Adaptive filter sizes / down-sampling rates based on the size of circle of -confusion: The first pass produces a buffer with circle of confusion sizes computed using the depth buffer, and the second pass blurs the image by either adapting the filter size or the down-sampling rate to dynamically change the size of the blurring kernel. Larger circle of confusion values result in larger blurs.
- Layered Depth of Field Methods: These methods use the depth buffer to separate the rendered image into many layers. Each layer is blurred with a different sized filter and then these layers are combined using sophisticated rules and alpha-blending. These methods were developed specifically to achieve the partial occlusion effect resulting from the “degeneration” of the screen-space projection of a given object due to its circle of confusion.

Arguably, the most important component in these depth of field pipelines is the blurring step. It should be noted that bokeh characteristics and shapes are entirely determined by the properties of the filter used to blur the image: for instance, if the blur is hexagonal shaped, the resulting blur will cause hexagonal-shaped bokeh. The blurring step is also, usually, the most time consuming step, as it requires multiple texture accesses (blurring is essentially writing a given fragment as the linear combination of surrounding fragments, which requires multiple texel fetches, given that the rendered image is supplied as a texture to the fragment shader) and therefore clever techniques must be employed to speed up this “critical-path”.

1.2 Filter Design

1.2.1 Filter Desiderata for Depth of Field Approximations

As of the time of the writing of this report, due to hardware, platform, and wrapper software limitations, filters for ***online*** depth of field approximation ***must*** satisfy:

Separability: The 2D convolution of an image and a filter takes significantly less time if the filter is a separable filter.

Desirable Shape: For a given fragment, the kernel must spread the fragment to nearby fragments in such a way that the resulting spread is in the shape of the bokeh.

"Good" Size: Convolution of an image with a large filter is computationally more expensive in comparison to the convolution of an image with a smaller filter. The filter size must be kept small for real-time applications; however, this means that to achieve larger blurs, the image itself must be down-sampled. If the sampling rate goes below the Nyquist frequency, this causes aliasing and thus is not desired. Therefore, the filter must be large enough to achieve a down-sampling rate that does not go below the Nyquist frequency while retaining the preferred size of the blur.

"Good" Energy Distribution: The convolution operation should distribute energy according a definition of "good" energy distribution function, e.g. a Gaussian distribution or a uniform distribution.

Filters ***should*** also satisfy:

Simplicity: The filtering process should not be too complex in terms of computational power and cognitive load.

Robustness/Efficaciousness: The filter should produce the intended visual effect in most cases, and should not produce undesirable artifacts.

Artistic Appeal: The filter should produce images that are deemed to be artistically pleasing/desired (e.g. hexagonal bokeh could be preferred over circular bokeh for a futuristic setting).

1.2.2 Motivation for Circularly Separable Filters in Depth Buffer Based Pipelines

1.2.2.1 Entry Point: Properties of Gaussian Filters

Gaussian filters are ubiquitously used for blurring and low-pass filtering due to the properties which have been discussed in the previous section. The purpose of this section is to demonstrate how to analyze a filter with regard to the properties discussed The general form of the filter is defined by the function

$$G_{\sigma_1,\sigma_2}(x,y) = \frac{1}{2\pi\sigma_1\sigma_2} e^{-\frac{x^2+y^2}{2\sigma_1\sigma_2}}$$

In most applications, the standard deviations σ_1 and σ_2, corresponding to the x and y dimensions are taken to be equal (i.e. $\sigma_1 = \sigma_2 = \sigma$), yielding

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Referring to the previous section, the Gaussian filter satisfies some of the "***must"*** criteria for filters used in depth of field approximation:

a) Gaussian filters are separable. A two-dimensional filter K_{xy} is said to be separable if and only if

$$K_{xy} = K_x K_y$$

where K_x and K_y are one dimensional column and row vectors, respectively and

$$I * K_{xy} = (I * K_x) * K_y$$

Notice that for a Gaussian, one can choose

$$K_{G_x} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

$$K_{G_y} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}$$

and obtain

$$K_{G_xG_y} = K_{G_x}K_{G_y} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = G_\sigma(x,y)$$

which implies that the Gaussian is separable.

b) Gaussian filters have desirable shape in the sense that they are isotropic/circularly symmetric filters. A 2D circular filter K can be defined as a filter having the property

$$K(x,y) = K(xcos\theta - ysin\theta, xsin\theta + ycos\theta)$$

i.e. for any rotation of the point $P(x,y)$ around the origin by θ the value of the filter should be the same for any values of x, y and θ.

Letting $C(x,y) = G\sigma(x,y)$, we observe that

$$G\sigma(xcos\theta - ysin\theta, xsin\theta + ysin\theta) = \frac{1}{2\pi\sigma^2} e^{-\frac{(xcos\theta - ysin\theta)^2 + (xsin\theta + ycos\theta)^2}{2\sigma^2}}$$

$$= \frac{1}{2\pi\sigma^2} e^{-\frac{x^2\cos^2\theta - 2xysin\theta cos\theta + y^2\sin^2\theta + x^2\sin^2\theta + 2xysin\theta cos\theta + y^2\cos^2\theta}{2\sigma^2}}$$

$$= \frac{1}{2\pi\sigma^2} e^{-\frac{x^2\cos^2\theta + y^2\sin^2\theta + x^2\sin^2\theta + y^2\cos^2\theta}{2\sigma^2}}$$

$$= \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)(\cos^2\theta + \sin^2\theta)}{2\sigma^2}}$$

$$= \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

$$= G\sigma(x,y)$$

which means that the Gaussian is isotropic.

c) It is possible to simply truncate a Gaussian to obtain a filter of “good” size while sufficiently maintaining its properties. Consider the following integral for a Gaussian distribution with $\mu = 0$ and σ, i.e. ($G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$):

$$\int_{-3\sigma}^{3\sigma} G(x')\, dx'$$

This integral is famously equal to 0.9973 as expressed by the 68–95–99.7 rule. This means that the Gaussian represents 99.73% of its data/energy within three standard deviations’ range. By making sure that the size of the filter is no less than 6σ, it is possible to satisfy the property of “good” size.

d) Energy distribution is satisfied if the desired distribution is a Gaussian distribution of energy. This is trivial to show, as the kernel for the convolution is literally a Gaussian. This is not satisfied if a uniform distribution is desired.

Gaussian filters are also simple and robust; however, for depth of field implementation, they lack the artistic appeal as the energy distribution of the filter is a Gaussian filter and hence, they do not produce visually appealing circular bokeh. Visually appealing bokeh distribute the energy uniformly while the Gaussian filter distributes the energy in a Gaussian manner, and therefore the attenuation of light intensity at the edges of bokeh generated by a Gaussian may not always be artistically appealing.

1.2.2.2 Uniform Circular Filters as a Means for Uniform Energy Distribution

It is possible to achieve a uniform distribution of energy by designing a uniform circular filter – a filter which is a circular region, and for every point within this region, the value of the filter is constant, i.e.

$$C(x,y) = \begin{cases} \frac{1}{\pi r^2}, & x^2 + y^2 \leq r^2 \\ 0, & x^2 + y^2 > r^2 \end{cases}$$

where r is the radius of the filter.

Convolution with this filter produces much better and accurate bokeh compared to the Gaussian; however, it does not satisfy an important property in the list of ***must*** have properties for a filter.

a) Uniform circular filters are NOT separable. Consider the singular value decomposition of a square matrix M of size rxr:

$$M = U\Sigma V^*$$

where U is a unitary matrix of size rxr, i.e. $U^*U = UU^* = I_{rxr}$, Σ is a diagonal matrix of size rxr the entries of which are the singular values of C in sorted order, and V is also a unitary matrix. Assume that M is separable, i.e. there exists a pair of column and row vectors M_x and M_y such that

$$M = M_x M_y$$

If Σ has a single entry σ, it is trivial to show that M is separable. Let u be the first column vector of U and let v be the first row vector of V^*. The matrix obtained by this decomposition will have entries

$$U\Sigma V^*_{i,j} = u_i \sigma v_j$$

where u_i denotes the i'th entry in u and v_j denotes the j'th entry in v. Choosing

$$M_x = \sigma^p u$$

$$M_y = \sigma^{1-p} v$$

where p is a positive real number we see that M is separable, referring to the definition of separability ($p = \frac{1}{2}$ is a nice choice). Notice that having more entries in Σ makes it impossible to choose M_x and M_y such that they conform to the definition of separability, since $U\Sigma V^*_{i,j}$ won't be in the form $u_i \sigma v_j$. Therefore, it is safe to argue that

$$If\ a\ matrix\ M\ is\ separable, then\ it\ has\ only\ one\ singular\ value.$$

Equivalently,

$$If\ a\ matrix\ M\ is\ separable, then\ rank(M) = 1.$$

Also, equivalently,

$$If\ rank(M) \neq 1, then\ M\ is\ not\ seperable.$$

Now, consider a matrix C_d of size axa where $a = 2r + 1$ formed by discretizing the function $C(x, y)$. C_d has a rank higher than one, therefore it is not separable.

b) Uniform circular filters have desirable shape in the sense that they are isotropic, i.e.

$$C(x, y) = C(xcos\theta - ysin\theta, xsin\theta + ycos\theta)$$

For all x, y, and θ

This is trivial to prove:

$C(xcos\theta - ysin\theta, xsin\theta + ycos\theta)$

$$= \begin{cases} \frac{1}{\pi r^2}, & (xcos\theta - ysin\theta)^2 + (xsin\theta + ycos\theta)^2 \leq r^2 \\ 0, & (xcos\theta - ysin\theta)^2 + (xsin\theta + ycos\theta)^2 > r^2 \end{cases}$$

$$= \begin{cases} \frac{1}{\pi r^2}, & (x^2 + y^2)(\cos^2\theta + \sin^2\theta) \leq r^2 \\ 0, & (x^2 + y^2)(\cos^2\theta + \sin^2\theta) > r^2 \end{cases}$$

$$= \begin{cases} \frac{1}{\pi r^2}, & x^2 + y^2 \leq r^2 \\ 0, & x^2 + y^2 > r^2 \end{cases}$$

$$= C(x, y)$$

c) For a uniform circular filter of size $(2r + 1)x(2r + 1)$, 100% percent of the energy of the filter is represented, therefore a uniform circular filter having such size will always have good coverage. This can be easily represented with the integral:

$$\int_{A^2} C(x, y)\, dx\, dy$$

$$= \int_{D^2} C(x,y)\,dx\,dy + \int_{A^2\setminus D^2} C(x,y)\,dx\,dy$$

$$= \int_{D^2} C(x,y)\,dx\,dy$$

$$= \int_0^{2\pi} \int_0^{r} \frac{1}{\pi r^2} r'dr'd\theta$$

$$= \frac{2\pi}{\pi r^2}\left[\frac{r'^2}{2}\right]_0^r$$

$$= 1$$

where A^2 is the region defined by the Cartesian product:

$$A^2 = [-2r-1, 2r+1]X[-2r-1, 2r+1]$$

And D^2 is the disk centered at the origin with a radius of r.

d) The energy is distributed uniformly over the coverage of the disk with a factor of $\frac{1}{\pi r^2}$. If the desired energy distribution is uniform, then the circular filter has a good energy distribution.

Uniform circular filters are simple and robust: constructing one is easy, and convolving an image with a circular filter is no different from convolving the image with another filter. For depth of field implementation, the uniform circular filter is a very good choice for simulating circular bokeh due to its uniform energy distribution and sharp fall-off at the edges of the bokeh. Therefore, it is strongly desired that a filter produces this effect.

1.2.3 Designing a Uniform Circularly Symmetric Separable Filter

We would like to combine the properties of two filters discussed in the previous section to accurately and quickly approximate the depth of field. The new filter should satisfy:

1. Separability
2. Circular Symmetricity
3. High energy coverage within a small size
4. Uniform distribution of energy within its coverage (i.e. a “good” energy distribution is a uniform distribution)

Note that neither the Gaussian filter nor the uniform circular filter satisfies all of these simultaneously. Gaussian filter fails to distribute energy uniformly, and uniform circular filter is not separable. The following sections provide the derivations for the nature of this new filter, henceforth referred to as uniform circularly symmetric separable filter.

1.2.3.1 Circularly Symmetric Separability and Uniformness

For the new filter, one needs to consider the family of functions which are both circularly symmetric and separable.

A 2D separable filter f' satisfies

$$f'(x,y) = f_1(x)f_2(y)$$

for a 1D pair of f_1 and f_2, and a 2D circularly symmetric filter satisfies

$$f'(x,y) = f'(xcos\theta - ysin\theta, xsin\theta + ycos\theta)$$

for all x, y, θ.

Combining these two definitions we get

$$f'(xcos\theta - ysin\theta, xsin\theta + ycos\theta) = f_1(x)f_2(y)$$

Noting that

$$(xcos\theta - ysin\theta)^2 + (xsin\theta + ycos\theta)^2 = x^2 + y^2$$

we can define a member of the family of circularly symmetric and separable filters, f, as

$$f\left(\sqrt{x^2+y^2}\right) = f(x)f(y)$$

Now that we have properly defined the family of circularly symmetric and separable filters, we need to solve the functional equation above for an explicit definition of this function. Substituting $g(x) = f(\sqrt{x})$, one obtains

$$g(x^2 + y^2) = g(x^2)g(y^2)$$

and further substituting $x^2 = u$ and $y^2 = v$ we obtain

$$g(u + v) = g(u)g(v)$$

This functional equation is called the exponential Cauchy functional equation [4]. The solution to this functional equation satisfies the properties below:

a) Letting $u = v = p/2$, we get $g(p) = \left(g\left(\frac{p}{2}\right)\right)^2$, and since the right-hand side is non-negative, we have that $g(p) \geq 0$.

b) If for some p_0, $g(p_0) = 0$, then the function vanishes everywhere. To see this, let $u = p_0$ and $v = p - p_0$. From the functional equation, we have $g(p - p_0 + p_0) = g(p_0)g(p - p_0)$ and thus $g(p) = g(p_0)g(p - p_0) = 0$. This also implies that if the function is 0 somewhere, it is zero everywhere, and hence if the function has a non-zero value, it does not have zero values.

c) Assuming that the function is non-zero, we have that $g(p) > 0$ for any p. This means that the logarithm function is defined over the range of this function. Taking the logarithm of both sides, one has

$$\log g(u + v) = \log g(u)g(v) = \log g(u) + \log g(v)$$

and substituting $h(x) = \log\ g(x)$ we have

$$h(u + v) = h(u) + h(v)$$

which is the linear Cauchy functional equation, which famously has the unique solution

$$h(x) = cx$$

for some constant c. Substituting back, one has that $g(x) = e^{cx}$, i.e. the nature of g is exponential.

We desire f to be non-zero, thus we choose g such that $g(x) > 0$, and substituting back we reach the conclusion

$$f(x) = e^{cx^2}$$

i.e. the family of circularly symmetric and separable filters are Gaussians.

This result, in itself, is not as interesting: we knew prior to this derivation that Gaussians are indeed circularly symmetric and separable as discussed previously. However, Gaussians do not uniformly distribute the energy and therefore this issue must be addressed as well.

To achieve uniformity in energy, we consider the generalization of a subset of the family of Gaussians to the complex domain: i.e. let G_c be a phased Gaussian such that

$$G_c(x, y) = e^{(-a+bi)(x^2+y^2)}$$

This filter is separable:

$$G_{c_x}(x) = e^{(-a+bi)x^2}$$

$$G_{c_y}(y) = e^{(-a+bi)y^2}$$

$$G_c(x, y) = G_{c_x}(x)G_{c_y}(y)$$

The output is in the complex domain and has imaginary components. To reduce the output to real numbers (required for practical purposes), a linear combination of the magnitude of the imaginary part and the real part of the filter is taken. Given an image I, we assign the output of the filter I'

$$I'(x, y) = ARe\{I(x, y) * G_c(x, y)\} + BIm\{I(x, y) * G_c(x, y)\}$$

where A and B are real valued constants.

This approach alone does not ensure uniform distribution, however. The model is extended one more time to consider the linear combination of multiple phased Gaussians to approximate a uniform filter. Consider the following Gaussians

$$G_{C_j}(x, y) = e^{(-a_j+b_ji)(x^2+y^2)}$$

where j ranges from 1 to N, a_j and b_j are real constants; and also consider the following output image

$$I'(x,y) = \sum_{j=1}^{N} A_j Re\{G_{c_j}(x,y) * I(x,y)\} + B_j Im\{G_{c_j}(x,y) * I(x,y)\}$$

where A_j and B_j are also real constants.

With this system, it is possible to achieve uniformity in filtering, by choosing the values of a_j, b_j, A_j, B_j such that

$$\sum_{j=1}^{N} A_j Re\{e^{(-a_j+b_j i)(x^2+y^2)}\} + B_j Im\left\{e^{(-a_j+b_j i)(x^2+y^2)}\right\} \cong \begin{cases} \frac{1}{\pi r^2}, & x^2+y^2 \leq r^2 \\ 0, & x^2+y^2 > r^2 \end{cases}$$

Equivalently,

$$\sum_{j=1}^{N} A_j e^{-a_j(x^2+y^2)} \cos\left(b_j(x^2+y^2)\right) + B_j e^{-a_j(x^2+y^2)} \sin\left(b_j(x^2+y^2)\right)$$

$$\cong \begin{cases} \frac{1}{\pi r^2}, & x^2+y^2 \leq r^2 \\ 0, & x^2+y^2 > r^2 \end{cases}$$

for some radius value r. This process resembles approximating a square wave using N terms by writing the Fourier series for the wave. What this approach does differently is that it takes the basis functions as the phased 2D Gaussians G_{c_j}.

The downside of this method is that it requires multiple passes over a render to produce uniformly circular bokeh: the sum of separable filters may not be separable, and in the case of this filter, the sum of its individual Gaussian filters is definitely not separable, as with very high probability the sum will have a rank more than 1. It is advantageous in the sense, however, that the individual passes over the render are separable, which should provide a significant speed up nevertheless.

1.2.3.2 Computing Filter Coefficients

Unfortunately, there exists no closed form expression that can be used to directly compute the coefficients a_j, b_j, A_j, B_j of these filters because anti-derivatives of Gaussians cannot be expressed in terms of elementary functions. Approaches resembling the analysis equation for Fourier Series do not also work since two Gaussians are not orthogonal. Also, given that the expression has many dimensions over which optimization should be performed (4 dimensions per filter component), it is better to make use of an algorithm that does not take a brute force approach. Hence it is more efficient to state this problem as an optimization problem, and then solve it using gradient descent or a genetic algorithm.

The following is the problem statement:

Consider the $4N$ dimensional space of row-wise concatenation of the vectors

$$S = \{Row - wise\ concatenation\ of\ (a_j, b_j, A_j, B_j)\ for\ all\ j\}$$

i.e.

$$S = (a_1, b_1, A_1, B_1, a_2, b_2, A_2, B_2, \dots, a_N, b_N, A_N, B_N)$$

Define the following functions corresponding to the uniform circular filter $C(x, y)$ and the proposed circularly symmetric separable uniform filter $F(x, y)$ as

$$C(x,y) = \begin{cases} \dfrac{1}{\pi r^2}, & x^2 + y^2 \leq r^2 \\ 0, & x^2 + y^2 > r^2 \end{cases}$$

$$F(S,x,y) = \sum_{j=1}^{N} A_j e^{-a_j(x^2+y^2)} \cos(b_j(x^2+y^2)) + B_j e^{-a_j(x^2+y^2)} \sin(b_j(x^2+y^2))$$

Define the error metric

$$E(S) = \iint_{R^2} \left(F(S,x,y) - C(x,y)\right)^2 dxdy$$

The set of optimal filter coefficients S_0 is hence given by

$$S_0 = \operatorname*{argmin}_S E(S)$$

For practical purposes, the outputs of C and F will be matrices and hence it is possible to change the error metric to

$$E(S) = \sum_k \sum_l \big(F(S,k,l) - C(k,l)\big)^2$$

where k, l are indices of the resulting matrix, so that we avoid computing an integral over an infinite domain.

Using a similar error metric, Niemitalo et al. compute the filter coefficients using a genetic optimization algorithm given in [5].

Garcia et al. provide a filter generator based on this method for GLSL and HLSL. The script is provided on GitHub [6]. Note that this script is bugged for Python 3, and it must be fixed by importing "reduce".

1.3 Implementation

The implementation was carried out in HLSL, and was glued to the game engine through the necessary classes and configuration files. The implementation makes use of the MRT technology and deferred rendering. The number of filter components was chosen to be two, as two components provide sufficient approximation and do not take as much time to blur the render. I can not provide the exact implementation in the report; however, I will provide the shader program pipeline. The very first node in the pipeline computes the size of the circle of confusion (CoC) for each fragment using the depth buffer. The following formula is used for computing CoC:

$$R_{CoC} = \min\left(max\left(\frac{R_m}{f_{far} - f}(depth - f), \frac{R_m}{f - f_{near}}(f - depth)\right), R_m\right)$$

where R_m is the maximum desired circle of confusion radius, f_{far} is the depth value for which the CoC radius value will attain its maximum value in the far plane, f_{near} is the depth value for which the CoC radius value will attain its maximum value in the near plane, f is the distance between the camera and the focal plane, and $depth$ is the depth buffer value for the fragment. Note that this expression is not a physically correct one. It was implemented in this way so that the artists could have more control over the placement of the blur at a particular depth value: the equation above simply allows the blur to vary linearly between $f_{near}, f,$ and f_{far} and is clamped between 0 and R_m. This way, the artists can simply designate a particular depth value as being in the depth of field and adjust the borders of the linearly varying blur by modifying f_{near} and f_{far}.

The output of this node is fed to two other nodes: the first node that receives the CoC radius buffer is the node that applies the horizontal blur for the first component. In this node, the following 1D convolution is computed for every fragment:

$$H_{1_{val}}(x,y) = \sum_{i=-R}^{R} K_1[i]I[x - \Delta r_x i, y]$$

$$H_{2_{val}}(x,y) = \sum_{i=-R}^{R} K_2[i]I[x - \Delta r_x i, y]$$

where R is the radius of the filter, K_1 and K_2 are the filter components, I is the rendered image, and Δr_x is the horizontal undersampling rate. The undersampling rate determines the size of the blur: the higher it is, the larger the blur will be at the cost of aliasing and resolution reduction. The undersampling rate allows one to artificially increase the size of the blur without having to increase the size of the filter. This way, the time required to compute the convolution above stays the same while the blur is larger. To compute I values at non-integer values, interpolation schemes such as bilinear interpolation or Lanczos resampling can be used. Δr_x can be chosen as the following:

$$\Delta r_x = \frac{1}{2} R_{CoC} V_x (1 + r)$$

where R_{CoC} is the size of the radius of the circle of confusion, V_x is the distance between two horizontally adjacent fragments, and r is an arbitrary adjustable parameter supplied to the shader. The output of the convolution are complex numbers, and one has to return four 16-bit floating-point buffers with four channels

(R16G16B16A16): two buffers for the real parts, two buffers for the imaginary parts. This is handled by returning a C-like struct with four 16-bit floating-point values with four channels (R16G16B16A16), which is then rendered with the MRT framework.

The second convolution node (vertical convolution node) takes in the CoC node output and the output of the first convolution node (horizontal convolution node), and performs the following vertical convolutions:

$$I_1(x,y) = \sum_{i=-R}^{R} K_1^T[i]H_{1_{val}}(x, y - \Delta r_y i)$$

$$I_2(x,y) = \sum_{i=-R}^{R} K_2^T[i]H_{2_{val}}(x, y - \Delta r_y i)$$

where Δr_y is the vertical undersampling rate, and can similarly be chosen as

$$\Delta r_y = \frac{1}{2} R_{CoC} V_y (1 + r)$$

where V_y is the distance between two vertically adjacent fragments. This node also combines the two images, I_1 and I_2 which are respectively the convolutions $K_1^T * K_1 * I$ and $K_2^T * K_2 * I$, in a linear fashion, as described in section 3.2.3.2:

$$I_{out} = w_{1,r}Re\{I_1\} + w_{1,i}Im\{I_1\} + w_{2,r}Re\{I_2\} + w_{2,i}Im\{I_2\}$$

This is returned in a single 16-bit-floating-point buffer with 4 channels (R16G16B16A16), and is piped to other post processing effects, which is beyond the scope of the work done.

1.4 Results and Quality-Performance Analysis

1.4.1 Theoretical Performance Analysis

Let K be the size of the kernel, M be the number of horizontal pixels/fragments, N be the number of vertical pixels/fragments and let L be the number of filter components.

Running on the CPU, no operations are parallelized. For every fragment, there will be $2K$ texel accesses in total (K for the vertical convolution component and K for the horizontal convolution component), and this convolution operation will be performed once per filter component, i.e. L times. The runtime complexity of this algorithm is $\Theta(KLMN)$. Assuming that $M \cong \mathrm{N}$ and noting that L is constant in practice, the complexity expression becomes $\Theta(KN^2)$.

Running on the GPU, the shader program containing instructions to perform this convolution is executed in parallel and hence the complexity becomes $\Theta(\mathrm{KL})$, and given that L is constant in practice (two component filters are more than sufficient for most purposes), the complexity becomes $\Theta(\mathrm{K})$, i.e. the time of computation depends on the radius of the filter components.

Compare the complexity of this algorithm to the uniform circular filter, which produces an almost identical output. Since the uniform circular filter is not separable, for every fragment, the number of texel accesses is K^2, thus resulting in an asymptotic runtime of $\Theta(K^2MN)$. Again, given that $M \cong \mathrm{N}$, the runtime becomes $\Theta(K^2N^2)$. Parallelizing the convolution on a GPU, the shader program is executed in parallel for every fragment, and thus the complexity of the convolution is $\Theta(K^2)$, which is asymptotically worse than the runtime of the previous algorithm.

In practice, however, one needs to consider the number of total operations performed by each algorithm. Convolution with a uniform circular filter performs K^2 texel fetches, $4K^2$ multiplications and $4(K^2-1)$ additions for every fragment (the factor 4 comes from having to multiply each color channel: r,g,b, and a). Assuming that the cost of a texel fetch is c_f, the cost of multiplication is c_m and cost of addition is c_a, the total cost of the algorithm when run in parallel is

$$c_{circular} = K^2\left(c_f + 4c_m + 4c_a\right) - 4c_a$$

Convolution with a uniform circularly symmetric separable filter performs $2K$ texel fetches per fragment and per filter component due to its separability; for the first filter component there are $16K$ multiplications ($2K$ texels with 4 real valued color channels each yield $8K$ complex number multiplications, and since complex number multiplication with a real number can be modelled as two real number multiplications, the total number is $16K$); for the rest of the components, there are $2K$ texels, 4 complex valued color channels, the complex multiplication of which can be modelled by 4 multiplications and 2 additions, thus yielding $32K$ multiplications and $16K$ additions. For the final gather, since there are L different buffers, there are $(L-1)$ additions performed in parallel for every fragment, and 2 multiplications and 1 addition for computing the linear combination of the real and the imaginary parts of the color channels for every filter component. In total, the cost is

$$c_{proposed} = 2KLc_f + 16Kc_m + (L-1)(32Kc_m + 16Kc_a + c_a) + 2Lc_m$$

In practice, the costs c_m and c_a are negligibly small. Substituting 0 for those expressions, we get the practical costs

$$c'_{circular} = K^2 c_f$$

$$c'_{proposed} = 2KLc_f$$

The ratio of the costs is

$$\frac{c'_{proposed}}{c'_{circular}} = \frac{2L}{K}$$

Since in practice K is taken around 17 and L is usually given to be 2, the theoretical ratio of execution times is $4/17$, meaning that the proposed algorithm is more than 4 times faster than convolving with a uniform circular filter.

1.4.2 Practical Performance Results

During my internship, I implemented both convolutions on the game engine as part of the post-processing framework. On an NVIDIA GeForce GTX780 at half resolution the depth of field implementation using the uniform circularly symmetric separable filter causes an overhead of around 1ms, whereas the naïve uniform circular filter takes around 3.5ms. The speedup is around 4 times, but not exactly 4 times. I am unable to provide further performance information about the implementation, since I do not have access to their engine anymore. However, I tried to recreate the implementation in MATLAB, separately from the game engine. The code and the resulting images are in the appendix and use the MATLAB Parallel Computing Toolbox. The implementation runs the convolutions on my GPU (NVIDIA GeForce GTX980). The following are the runtimes of the algorithms on a single image, averaged over 100 iterations:

Separable Gaussian: Average Elapsed time is 0.096654 seconds.
Uniform Circular Filter: Average Elapsed time is 0.155347 seconds.
Uniform Circularly Symmetric Separable Filter: Average Elapsed time is 0.102996 seconds.

Unfortunately, the theoretical discussion fails to explain practical results. While the proposed filter is faster than the regular circular filter, it is not 4 times faster. This could be explained by the communication costs between the CPU and the GPU: the circularly symmetric separable filter requires more data to be transferred between the two units, and hence suffers some overhead due to this transfer.

References

[1] M. Pharr, J. Wenzel, and G. Humphreys, "6.4 Realistic Cameras" in *Physically based rendering: from theory to implementation*. Amsterdam: Morgan Kaufmann, 2017. [Accessed: Oct 17, 2019].

[2] P. Haeberli and K. Akeley, "The accumulation buffer: hardware support for high-quality rendering," *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 4, pp. 309–318, Jan. 1990.

[3] "GPU Gems," NVIDIA Developer. [Online]. Available: https://developer.nvidia.com/sites/all/modules/custom/gpugems/books/GPUGems/gpugems_ch23.html. [Accessed: 17-Oct-2019].

[4] C. Efthimiou, “5.2 Second Cauchy Equation” in *Introduction to functional equations: theory and problem-solving strategies for mathematical competitions and beyond*. Berkeley: Mathematical Sciences Research Institute, 2011.

[5] O. Niemitalo, “Evolutionary real variable optimization in C,” ikifio, 05-Jun-2019. [Online]. Available: http://yehar.com/blog/?p=643. [Accessed: 17-Oct-2019].

[6] Kecho, “kecho/CircularDofFilterGenerator,” GitHub, 17-Oct-2019. [Online]. Available: https://github.com/kecho/CircularDofFilterGenerator. [Accessed: 17-Oct-2019].

Appendices

Contents of sep_circ_conv.m:

```
WK0 = [-22.356787 85.912460];
RK0 = [-7.5977237626584365e-06 -4.617458623370157e-06 -
4.243185905711808e-07 5.238664624631662e-06 1.2661511372866501e-05
2.2168599478214192e-05 3.4119538884911e-05 4.89095337747343e-05
6.696915200290912e-05 8.876342573073745e-05 0.0001147902137355905
0.00014557776513569008 0.00018168143617845463 0.00022367952621741286
0.00027216821585898253 0.0003277556092154783 0.0003910549028787102
0.0004626767261656398 0.0005432207198419041 0.0006332664432951556
0.0007333637223566974 0.0008440225709800646 0.0009657028391002589
0.0010988037555604803 0.0012436535483949728 0.001400499334458443 7
0.0015694974759500268 0.0017507046024614645 0.0019440694935834691
0.0021494260087695912 0.002366487238169241 0.002594841030733221
0.002833947034441033 0.003083135358504137 0.003341606939482618 4
0.0036084356631260733 0.003882572262192361 0.004162849978329325
0.004447991944153493 0.004736620210735469 0.005027266316564921
0.005318383267404137 0.005608358772836323 0.005895529565244219
0.006178196610759253 0.006454641009606975 5 0.006723140375310554
0.0069819854783370145 0.007229496939796574 0.007464041764443449
0.0076840495091012855 0.007888027892308965 0.008074577662976251
0.008242406559663634 0.008390342207277874 0.008517343814059043
0.008622512548332455 0.008705100491283738 0.008764518078736776
0.008800339961403766 0.008812309229244544 0.008800339961403766
0.008764518078736776 0.008705100491283738 0.008622512548332455
0.008517343814059043 0.008390342207277874 0.008242406559663634
0.008074577662976251 0.007888027892308965 0.0076840495091012855
0.007464041764443449 0.007229496939796574 0.0069819854783370145
0.006723140375310554 0.006454641009606975 5 0.006178196610759253
0.005895529565244219 0.005608358772836323 0.005318383267404137
```

```
0.005027266316564921 0.004736620210735469 0.004447991944153493
0.004162849978329325 0.003882572262192361 0.0036084356631260733
0.0033416069394826184 0.0030831353585504137 0.002833947034441033
0.002594841030733221 0.002366487238169241 0.0021494260087695912
0.0019440694935834691 0.0017507046024614645 0.0015694974759500268
0.0014004993344584437 0.0012436535483949728 0.0010988037555604803
0.0009657028391002589 0.0008440225709800646 0.0007333637223566974
0.0006332664432951556 0.0005432207198419041 0.0004626767261656398
0.0003910549028787102 0.0003277556092154783 0.00027216821585898253
0.00022367952621741286 0.00018168143617845463 0.00014557776513569008
0.0001147902137355905 8.876342573073745e-05 6.696915200290912e-05
4.89095337747343e-05 3.4119538884911e-05 2.2168599478214192e-05
1.2661511372866501e-05 5.238664624631662e-06 -4.243185905711808e-07 -
4.617458623370157e-06 -7.5977237626584365e-06 ];
IK0 = [6.567041732438569e-05 7.757586651456923e-05
9.110599319078506e-05 0.00010639039037257059 0.00012355373231083894
0.00014271268995911756 0.00016397255712825444 0.00018742363075110755
0.00021313740182291334 0.00024116262673746095 0.000271521361463943
0.000304205052820326 0.00033917079148730746 0.0003763378398557237
0.0004155845538041958 0.00045674582058962006 0.0004996111347808883
0.00054392343023106 0.0005893787782151397 0.0006356270499232186
0.0006822736254843395 0.0007288822117375781 0.00077497880734326
0.0008200568269674045 0.0008635833667496587 0.000905006561787005
0.0009437639537592736 0.0009792917540121689 0.0010110348553943 82
0.0010384574159528648 0.0010610538102662334 0.0010783597207525912
0.0010899631226703207 0.0010955149035816254 0.0010947388514753061
0.0010874407460870626 0.0010735162955594622 0.0010529576755843072
0.0010258584504768237 0.0009924166849224977 0.0009529360908640125
0.0009078250953900578 0.0008575937615827922 0.0008028485439350246
0.000744284911877971 0.0006826779277767077 0.0006188709180004186
0.0005537624258857348 0.0004882916821373003 0.00042342287007455067
0.00036012849887640306 0.00029937222648961174 0.00024209149423580283
0.00018918034668071574 0.00014147281256464216 9.972721534250743e-05
6.461176521989778e-05 3.6691758839177325e-05 1.6418678568053e-05
4.1214415145938045e-06 0.0 4.1214415145938045e-06 1.6418678568053e-05
3.6691758839177325e-05 6.461176521989778e-05 9.972721534250743e-05
0.00014147281256464216 0.00018918034668071574 0.00024209149423580283
0.00029937222648961174 0.00036012849887640306 0.00042342287007455067
0.0004882916821373003 0.0005537624258857348 0.0006188709180004186
0.0006826779277767077 0.000744284911877971 0.0008028485439350246
0.0008575937615827922 0.0009078250953900578 0.0009529360908640125
0.0009924166849224977 0.0010258584504768237 0.0010529576755843072
0.0010735162955594622 0.0010874407460870626 0.0010947388514753061
0.0010955149035816254 0.0010899631226703207 0.0010783597207525912
0.0010610538102662334 0.0010384574159528648 0.0010110348553943 82
0.0009792917540121689 0.0009437639537592736 0.000905006561787005
0.0008635833667496587 0.0008200568269674045 0.00077497880734326
0.0007288822117375781 0.0006822736254843395 0.0006356270499232186
0.0005893787782151397 0.00054392343023106 0.0004996111347808883
0.00045674582058962006 0.0004155845538041958 0.0003763378398557237
0.00033917079148730746 0.000304205052820326 0.000271521361463943
0.00024116262673746095 0.00021313740182291334 0.00018742363075110755
0.00016397255712825444 0.00014271268995911756 0.00012355373231083894
0.00010639039037257059 9.110599319078506e-05 7.757586651456923e-05
6.567041732438569e-05 ];
WK1 = [35.918936 -28.875618];
RK1 = [2.2353103356900706e-05 1.1160560940980838e-05 -
4.483085304828264e-06 -2.5064623405942437e-05 -5.096051962626105e-05
-8.239996010489192e-05 -0.00011942886936089635 -
0.00016187667177949743 -0.00020932758768104457 -
0.00026109818740906605 -0.00031622276365377824 -
```

```
0.00037344782827304537 -0.000431236705115124 -0.00048778478983245216
-0.0005410456007160743 -0.0005887672736177456 -0.0006285386832824777
-0.0006578439274253925 -0.0006741235120292934 -0.0006748402474399871
-0.0006575476219637221 -0.0006199582751837049 -0.0005600101541195396
-0.00047592800301826026 -0.0003662780077624793 -
0.00023001367918323965 -6.6511402027518e-05 0.00012440551958972952
0.00034245504377563175 0.0005868970571974311 0.0008565463194044071
0.0011497970357876915 0.001464658007677916 0.0017987969620634344
0.0021495923857902267 0.0025141909882970092 0.002889568797371242
0.0032725938545871847 0.003660088517859463 0.004048889491556941
0.0044359038808676095 0.004818159795582306 0.00519285029685527
0.005557369775851457 0.005909342162559039 0.0062466406740603435
0.006567399112894998 0.006870015007910261 0.00715314514393901
0.0074156942463458045 0.007656797767347355 0.007875799860239759
0.008072227724094055 0.008245763555416428 0.008396215356244942
0.00852348782269597 0.008627554477352502 0.008708432116896236
0.008766158527140902 0.008800774275401125 0.008812309229244544
0.008800774275401125 0.008766158527140902 0.008708432116896236
0.008627554477352502 0.00852348782269597 0.008396215356244942
0.008245763555416428 0.008072227724094055 0.007875799860239759
0.007656797767347355 0.0074156942463458045 0.00715314514393901
0.006870015007910261 0.006567399112894998 0.0062466406740603435
0.005909342162559039 0.005557369775851457 0.00519285029685527
0.004818159795582306 0.0044359038808676095 0.004048889491556941
0.003660088517859463 0.0032725938545871847 0.002889568797371242
0.0025141909882970092 0.0021495923857902267 0.0017987969620634344
0.001464658007677916 0.0011497970357876915 0.0008565463194044071
0.0005868970571974311 0.00034245504377563175 0.00012440551958972952 -
6.6511402027518e-05 -0.00023001367918323965 -0.0003662780077624793 -
0.00047592800301826026 -0.0005600101541195396 -0.0006199582751837049
-0.0006575476219637221 -0.0006748402474399871 -0.0006741235120292934
-0.0006578439274253925 -0.0006285386832824777 -0.0005887672736177456
-0.0005410456007160743 -0.00048778478983245216 -0.000431236705115124
-0.00037344782827304537 -0.00031622276365377824 -
0.00026109818740906605 -0.00020932758768104457 -
0.00016187667177949743 -0.00011942886936089635 -8.239996010489192e-05
-5.096051962626105e-05 -2.5064623405942437e-05 -4.483085304828264e-06
1.1160560940980838e-05 2.2353103356900706e-05 ];
IK1 = [-7.598495058991654e-05 -9.18782707613468e-05 -
0.00010777650483986324 -0.0001228610321408201 -0.00013615436043025906
-0.00014653188326750136 -0.0001527416460471059 -
0.00015343211400891107 -0.00014718753570753076 -0.0001325700879535787
-0.00010816759585946327 -7.264526719104367e-05 -2.4799585166460278e-
05 3.638771324499525e-05 0.0001116977710375465 0.000201627076919768
0.0003063463996929454 0.00042566814051461376 0.0005590237689113108
0.0007054526421127473 0.0008636030757549343 0.0010317460583286228
0.0012078015051686822 0.0013893764543077883 0.0015738141395782506
0.0017582524574765025 0.00193968999202622 0.002115057490785454
0.0022812925052809385 0.002435414825661791 0.0025746003524519724
0.0026962511534584224 0.0027980596424726366 0.002878065076196884
0.002934700881984238 0.0029668316849307667 0.002973779281258688
0.002955337188602796 0.0029117737765881864 0.0028438243285261225
0.002752672695021469 0.002639923463350032 0.002507565776130447
0.002357930085573801 0.0021936392248216603 0.0020175552176039664
0.001832723235987477 0.0016423140595403169 0.001449566295401391
0.0012577294960036754 0.0010700091684507577 0.0008895145156201283
0.0007192095923234523 0.0005618684078234129 0.00042003436510713497
0.00029598430264691323 0.00019169729958945497 0.00010882832254777458
4.868673207346508e-05 1.2219628689350604e-05 0.0 1.2219628689350604e-
05 4.868673207346508e-05 0.00010882832254777458
0.00019169729958945497 0.00029598430264691323 0.00042003436510713497
```

```
0.0005618684078234129 0.0007192095923234523 0.0008895145156201283
0.0010700091684507577 0.0012577294960036754 0.001449566295401391
0.0016423140595403169 0.001832723235987477 0.0020175552176039664
0.0021936392248216603 0.002357930085573801 0.002507565776130447
0.0026399234633350032 0.002752672695021469 0.0028438243285261225
0.0029117737765881864 0.002955337188602796 0.002973779281258688
0.0029668316849307667 0.002934700881984238 0.002878065076196884
0.0027980596424726366 0.0026962511534584224 0.0025746003524519724
0.002435414825661791 0.0022812925052809385 0.002115057490785454
0.00193968999202622 0.0017582524574765025 0.0015738141395782506
0.0013893764543077883 0.0012078015051686822 0.0010317460583286228
0.0008636030757549343 0.0007054526421127473 0.0005590237689113108
0.00042566814051461376 0.0003063463996929454 0.000201627076919768
0.0001116977710375465 3.638771324499525e-05 -2.4799585166460278e-05 -
7.264526719104367e-05 -0.00010816759585946327 -0.0001325700879535787
-0.00014718753570753076 -0.00015343211400891107 -
0.0001527416460471059 -0.00014653188326750136 -0.00013615436043025906
-0.0001228610321408201 -0.00010777650483986324 -9.18782707613468e-05
-7.598495058991654e-05 ];
WK2 = [-13.212253 -1.578428];
RK2 = [-5.823431255990815e-05 -2.0550887156950554e-05
2.9894356666588122e-05 9.135671277133185e-05 0.00016078036471860654
0.0002338684365446267 0.0003052654102607885 0.0003688438990885145
0.000418078166685033 0.0004464786851040497 0.0004480561949629971
0.0004177806876205226 0.00035200069637821116 0.0002487911869363825
0.00010820382610350005 -6.759909217483164e-05 -0.00027433700015713823
-0.0005057303565551289 -0.0007537385976159024 -0.0010089004937845487
-0.0012607543014295857 -0.0014983107090982824 -0.0017105493583144237
-0.0018869096716076766 -0.002017748672883917 -0.0020947421242519504 -
0.0021112102168075802 -0.002062354774605742 -0.001945400985338562 -
0.0017596426116294579 -0.0015063950775441055 -0.0011888654626603262 -
0.0008119520621640556 -0.0003819886744880062 9.35498617788066e-05
0.0006063640825431163 0.0011476521468919093 0.0017084232741494672
0.002279789790619379 0.00285322766568363 0.0034207980697036617
0.00397532515531995 0.004510527798886838 0.0050211053223581294
0.005502779165208222 0.005952294042184305 0.0063673832881129605
0.006746703864422767 0.007089746913384063 0.007396729840526454
0.007668475738161977 0.007906285592789155 0.008111808205765854
0.00828691215579988 0.008433563492983773 0.00855371221861477
0.008649190005032647 0.008721621067797945 0.008772347632045832
0.008802371040478707 0.008812309229244544 0.008802371040478707
0.008772347632045832 0.008721621067797945 0.008649190005032647
0.00855371221861477 0.008433563492983773 0.00828691215579988
0.008111808205765854 0.007906285592789155 0.007668475738161977
0.007396729840526454 0.007089746913384063 0.006746703864422767
0.0063673832881129605 0.005952294042184305 0.005502779165208222
0.0050211053223581294 0.004510527798886838 0.00397532515531995
0.0034207980697036617 0.00285322766568363 0.002279789790619379
0.0017084232741494672 0.0011476521468919093 0.0006063640825431163
9.35498617788066e-05 -0.0003819886744880062 -0.0008119520621640556 -
0.0011888654626603262 -0.0015063950775441055 -0.0017596426116294579 -
0.001945400985338562 -0.002062354774605742 -0.0021112102168075802 -
0.0020947421242519504 -0.002017748672883917 -0.0018869096716076766 -
0.0017105493583144237 -0.0014983107090982824 -0.0012607543014295857 -
0.0010089004937845487 -0.0007537385976159024 -0.0005057303565551289 -
0.00027433700015713823 -6.759909217483164e-05 0.00010820382610350005
0.0002487911869363825 0.00035200069637821116 0.0004177806876205226
0.0004480561949629971 0.0004464786851040497 0.000418078166685033
0.0003688438990885145 0.0003052654102607885 0.0002338684365446267
0.00016078036471860654 9.135671277133185e-05 2.9894356666588122e-05 -
2.0550887156950554e-05 -5.823431255990815e-05 ];
```

```
IK2 = [0.00014159534581737008 0.000173840453236714
0.00019744509608821877 0.00020812941751392785 0.000201944708677025
0.0001756589932420594 0.0001271221500828562 5.557772373927735e-05 -
3.810795307893025e-05 -0.0001513224612843317 -0.0002797045795627203 -
0.0004172672141922628 -0.0005566409507719142 -0.0006894236504254931 -
0.0008066133264844397 -0.0008990953828427079 -0.0009581516140609869 -
0.000975957352572147 -0.0009460347442222877 -0.0008636340608500465 -
0.0007260207599532558 -0.0005326530975562474 -0.0002852428533043859
1.2300498588480163e-05 0.00035403462082276763 0.000732244412047775
0.0011377788402228877 0.0015604427709819087 0.0019894205030055413
0.002413707766379233 0.002822530403343376 0.0032057305796765073
0.0035541048644841255 0.003859682540905352 0.00411593675096487
0.004317925243913363 0.0044623613436658785 0.004547619087973987
0.0045736791908260735 0.0045420244711323506 0.004455494660750283
0.004318111085878354 0.004134881677148228 0.003911596200764631
0.0036546206265654804 0.0033706982753754617 0.003066763931404235
0.002749775570528995 0.0024265668332506853 0.0021037219365999484
0.0017874734293457121 0.0014836220891838972 0.001197477362712566
0.0009338160684244438 0.0006968566175649291 0.0004902457464455643
0.0003170546791996695 0.00017978173046766138 8.035858943049388e-05
2.0157875698098586e-05 0.0 2.0157875698098586e-05 8.035858943049388e-
05 0.00017978173046766138 0.0003170546791996695 0.0004902457464455643
0.0006968566175649291 0.0009338160684244438 0.001197477362712566
0.0014836220891838972 0.0017874734293457121 0.0021037219365999484
0.0024265668332506853 0.002749775570528995 0.003066763931404235
0.0033706982753754617 0.0036546206265654804 0.003911596200764631
0.004134881677148228 0.004318111085878354 0.004455494660750283
0.0045420244711323506 0.0045736791908260735 0.004547619087973987
0.0044623613436658785 0.004317925243913363 0.00411593675096487
0.003859682540905352 0.0035541048644841255 0.0032057305796765073
0.002822530403343376 0.002413707766379233 0.0019894205030055413
0.0015604427709819087 0.0011377788402228877 0.000732244412047775
0.00035403462082276763 1.2300498588480163e-05 -0.0002852428533043859
-0.0005326530975562474 -0.0007260207599532558 -0.0008636340608500465
-0.0009460347442222877 -0.000975957352572147 -0.0009581516140609869 -
0.0008990953828427079 -0.0008066133264844397 -0.0006894236504254931 -
0.0005566409507719142 -0.0004172672141922628 -0.0002797045795627203 -
0.0001513224612843317 -3.810795307893025e-05 5.557772373927735e-05
0.0001271221500828562 0.0001756589932420594 0.000201944708677025
0.00020812941751392785 0.00019744509608821877 0.000173840453236714
0.00014159534581737008 ];
WK3 = [0.507991 1.816328];
RK3 = [0.0003704278294488616 0.0002541242853045671
7.120797578134319e-05 -0.00015808395553878465 -0.00040392715795944224
-0.000630854968514271 -0.0008026897140968201 -0.0008877069343732477 -
0.0008632684968613246 -0.0007192699019108036 -0.0004599485499599577 -
0.00010385366330205109 0.00031795860705828414 0.0007652093471836394
0.0011926937410513053 0.0015550312145977622 0.0018112983990614393
0.0019290366645517197 0.0018872439922428916 0.00167811231156706
0.0013074326524059952 0.000793740467520184 0.00016639600056466488 -
0.0005371205497516941 -0.0012743759406144 -0.0020012860937773775 -
0.002675256809455541 -0.003257920972516586 -0.0037172998328620214 -
0.004029290634292566 -0.0041784542665065086 -0.004158138332608875 -
0.003970018952442639 -0.0036231768906967825 -0.0031328402854917067 -
0.002518928940846366 -0.0018045264203618413 -0.0010143891360945533 -
0.0001735794470949423 0.0006937145984391751 0.001565134359223892
0.0024208639912849154 0.0032441041645909887 0.004021322639942601
0.004742308491182864 0.005400062968910694 0.005990562656116234
0.006512430308089653 0.006966546309361964 0.0073556297210400495
0.007683813071481461 0.007956229904086039 0.008178629067741392
0.008357025133020031 0.008497390348390829 0.00860539033023793
```

```
0.00868616324724428 0.00874414059443613 0.008782906694078293
0.0088050937214745 0.008812309229244544 0.0088050937214745
0.008782906694078293 0.00874414059443613 0.00868616324724428
0.00860539033023793 0.008497390348390829 0.008357025133020031
0.008178629067741392 0.007956229904086039 0.007683813071481461
0.0073556297210400495 0.006966546309361964 0.006512430308089653
0.005990562656116234 0.005400062968910694 0.004742308491182864
0.004021322639942601 0.0032441041645909887 0.0024208639912849154
0.001565134359223892 0.0006937145984391751 -0.0001735794470949423 -
0.0010143891360945533 -0.0018045264203618413 -0.002518928940846366 -
0.0031328402854917067 -0.0036231768906967825 -0.003970018952442639 -
0.004158138332608875 -0.0041784542665065086 -0.004029290634292566 -
0.0037172998328620214 -0.003257920972516586 -0.002675256809455541 -
0.0020012860937773775 -0.0012743759406144 -0.0005371205497516941
0.00016639600056466488 0.000793740467520184 0.0013074326524059952
0.00167811231156706 0.0018872439922428916 0.0019290366645517197
0.0018112983990614393 0.0015550312145977622 0.0011926937410513053
0.0007652093471836394 0.00031795860705828414 -0.00010385366330205109
-0.0004599485499599577 -0.0007192699019108036 -0.0008632684968613246
-0.0008877069343732477 -0.0008026897140968201 -0.000630854968514271 -
0.00040392715795944224 -0.00015808395553878465 7.120797578134319e-05
0.0002541242853045671 0.0003704278294488616 ];
IK3 = [-0.0002907763692770937 -0.00045229953262560603 -
0.0005661568975541097 -0.0006063195134882596 -0.0005556246715244903 -
0.0004090461604896031 -0.0001752184356257717 0.00012401306623637732
0.00045593233855508564 0.0007808219316910395 0.00105679353108688
0.0012449062352930904 0.00131389762678773 0.0012439463349387882
0.001029049508007764 0.0006778032770395397 0.00021258610719964994 -
0.0003326653923120503 -0.0009157531200455144 -0.00149016296215475 -
0.002009207149903248 -0.0024299384079511308 -0.0027164815910830547 -
0.002842528346318619 -0.0027928533428734264 -0.0025638220819011936 -
0.0021629583197712568 -0.0016077155037884005 -0.0009236470704834412 -
0.00014219439067515317 0.0007016890681948187 0.001571878700855933
0.002433220187798724 0.0032533489412375953 0.0040004110069901492
0.004662544429081538 0.0052114440920463575 0.005639510860718454
0.005941173212575823 0.0061161303386303035 0.006168697406616469
0.00610702513450703 0.005942260700553968 0.005687707529273279
0.005358030066170914 0.004968537574070685 0.004534569275378126
0.004070992570271758 0.0035918170522814593 0.003109919852557813
0.0026368725102415384 0.0021828559810228598 0.0017566483540306414
0.001365669088905911 0.0010160638317811641 0.000712814851001831
0.0004598636087329833 0.0002602337507203965 0.00011614469633456949
2.9107933494648952e-05 0.0 2.9107933494648952e-05
0.00011614469633456949 0.0002602337507203965 0.0004598636087329833
0.000712814851001831 0.0010160638317811641 0.001365669088905911
0.0017566483540306414 0.0021828559810228598 0.0026368725102415384
0.003109919852557813 0.0035918170522814593 0.004070992570271758
0.004534569275378126 0.004968537574070685 0.005358030066170914
0.005687707529273279 0.005942260700553968 0.00610702513450703
0.006168697406616469 0.0061161303386303035 0.005941173212575823
0.005639510860718454 0.0052114440920463575 0.004662544429081538
0.0040004110069901492 0.0032533489412375953 0.002433220187798724
0.001571878700855933 0.0007016890681948187 -0.00014219439067515317 -
0.0009236470704834412 -0.0016077155037884005 -0.0021629583197712568 -
0.0025638220819011936 -0.0027928533428734264 -0.002842528346318619 -
0.0027164815910830547 -0.0024299384079511308 -0.002009207149903248 -
0.00149016296215475 -0.0009157531200455144 -0.0003326653923120503
0.00021258610719964994 0.0006778032770395397 0.001029049508007764
0.0012439463349387882 0.00131389762678773 0.0012449062352930904
0.00105679353108688 0.0007808219316910395 0.00045593233855508564
0.00012401306623637732 -0.0001752184356257717 -0.0004090461604896031
```

```
-0.0005556246715244903 -0.0006063195134882596 -0.0005661568975541097
-0.0004522995326256060 3 -0.0002907763692770937 ];
WK4 = [0.138051 -0.010000];
RK4 = [-0.0017813263838037125 -0.0020247635196323746 -
0.0017087506814179503 -0.000892265681412454 0.00023081927991513518
0.0013851758675104335 0.0022854961298344393 0.0027062941018057697
0.0025330076923384755 0.001783995758954127 0.0006012348366393512 -
0.0007849849459034136 -0.0021069628777637952 -0.0031137411635800455 -
0.0036172305887041674 -0.0035223106367041155 -0.00283726480996271 2 -
0.0016652235345800217 -0.00018068380253728598 0.0014028275097264422
0.0028673718812035104 0.004021682482565417 0.004724506137381442
0.004897984287134677 0.004530788784952383 0.0036722989295041907
0.0024201402407567634 0.0009038658549746074 -0.0007324848210319253 -
0.002346637213705394 -0.00381147103839985 1 -0.005024590616670278 -
0.0059137864313505 -0.006438640944861548 -0.006588909822659612 -
0.0063805179475138725 -0.005850067294230612 -0.005048694673094713 -
0.0040359807537429125 -0.002874435257716829 -0.0016248982370480323 -
0.00034302707916996 5 0.0009231022512093734 0.0021343565086657506
0.0032610146373898 67 0.004282527129898672 0.005186756370329723
0.00596887877252876 0.006630106445031744 0.007176356257102856
0.007616962326089861 0.007963497553470276 0.008228743120498646
0.008425822981625752 0.008567503724318115 0.008665648491971763
0.008730806438575631 0.008771915674048105 0.008796097101013145
0.008808518216424446 0.008812309229244544 0.008808518216424446
0.008796097101013145 0.008771915674048105 0.008730806438575631
0.008665648491971763 0.008567503724318115 0.008425822981625752
0.008228743120498646 0.007963497553470276 0.007616962326089861
0.007176356257102856 0.006630106445031744 0.00596887877252876
0.005186756370329723 0.004282527129898672 0.003261014637389867
0.0021343565086657506 0.0009231022512093734 -0.000343027079169965 -
0.0016248982370480323 -0.002874435257716829 -0.0040359807537429125 -
0.005048694673094713 -0.005850067294230612 -0.0063805179475138725 -
0.006588909822659612 -0.006438640944861548 -0.0059137864313505 -
0.005024590616670278 -0.003811471038399851 -0.002346637213705394 -
0.0007324848210319253 0.0009038658549746074 0.0024201402407567634
0.0036722989295041907 0.004530788784952383 0.004897984287134677
0.004724506137381442 0.004021682482565417 0.0028673718812035104
0.0014028275097264422 -0.00018068380253728598 -0.0016652235345800217
-0.002837264809962712 -0.0035223106367041155 -0.0036172305887041674 -
0.0031137411635800455 -0.0021069628777637952 -0.0007849849459034136
0.0006012348366393512 0.001783995758954127 0.0025330076923384755
0.0027062941018057697 0.0022854961298344393 0.0013851758675104335
0.00023081927991513518 -0.000892265681412454 -0.0017087506814179503 -
0.0020247635196323746 -0.0017813263838037125 ];
IK4 = [-0.0007708734778948588 0.00025301777821438683
0.0012938833227578422 0.0020649585939261947 0.002347572787146532
0.0020469012710433564 0.0012128949095992604 2.4043669848744997e-05 -
0.001258690965573542 -0.002353995350911606 -0.0030224834351417116 -
0.0031152468124251414 -0.002599694925412647 -0.0015598511700210 46 -
0.00017402606423710511 0.001323046160137356 0.002684535401581128
0.003693414914807718 0.004194909995028 77 0.004115318791919236
0.0034660938524983 15 0.0023347385059319554 0.000865938781633988 -
0.0007627343968523162 -0.002366198854825491 6 -0.0037746298272391646 -
0.004851036017481358 -0.005502294682325835 -0.005683455075377575 -
0.005396040710574592 -0.004681688630471789 -0.003612750014078795 -
0.0022814832399629483 -0.0007892693993031 86 0.0007630474582297305
0.0022829712230348427 0.003691958838255219 5 0.004928776564504219
0.005950801715244426 0.006733618400099588 0.007269332495365031
0.007564043575480188 0.00763487686471062 0.007506913727412815
0.007210280240805911 0.006777572279733808 0.006241721007157903
0.005634340004229259 0.00498454688241159 0.004318218272747316 5
```

```
0.0036576162539287815 0.0030213143658256495 0.0024243498144345134
0.0018785327920202706 0.0013928517684264883 0.0009739233146214997
0.0006264451032352098 0.00035362023150286963 0.00015752935056259477
3.943399800295202e-05 0.0 3.943399800295202e-05
0.00015752935056259477 0.00035362023150286963 0.0006264451032352098
0.0009739233146214997 0.0013928517684264883 0.0018785327920202706
0.0024243498144345134 0.0030213143658256495 0.0036576162539287815
0.0043182182727473165 0.004984546882411159 0.005634340004229259
0.006241721007157903 0.006777572279733808 0.007210280240805911
0.007506913727412815 0.00763487686471062 0.007564043575480188
0.007269332495365031 0.006733618400099588 0.005950801715244426
0.004928776564504219 0.0036919588382552195 0.0022829712230348427
0.0007630474582297305 -0.000789269399303186 -0.0022814832399629483 -
0.003612750014078795 -0.004681688630471789 -0.005396040710574592 -
0.005683455075377575 -0.005502294682325835 -0.004851036017481358 -
0.0037746298272391646 -0.0023661988548254916 -0.0007627343968523162
0.000865938781633988 0.0023347385059319554 0.003466093852498315
0.004115318791919236 0.00419490999502877 0.003693414914807718
0.002684535401581128 0.001323046160137356 -0.00017402606423710511 -
0.001559851170021046 -0.002599694925412647 -0.0031152468124251414 -
0.0030224834351417116 -0.002353995350911606 -0.001258690965573542
2.4043669848744997e-05 0.0012128949095992604 0.0020469012710433564
0.002347572787146532 0.0020649585939261947 0.0012938833227578422
0.00025301777821438683 -0.0007708734778948588 ];
RADIUS = 60;


%{
WK0  = [0.411259 -0.548794];
RK0 = [0.01409556159668803 -0.020612459315238564 -0.03870755434408963
-0.021448763206040992 0.013015130310381085 0.04217799580303998
0.057972226428136375 0.0636473791315247 0.0647544760221452
0.0636473791315247 0.057972226428136375 0.04217799580303998
0.013015130310381085 -0.021448763206040992 -0.03870755434408963 -
0.020612459315238564 0.01409556159668803 ];
IK0 = [-0.022657586462979884 -0.025574203141729788
0.006956906412901003 0.04046812518812997 0.05022292778495918
0.03858507697191389 0.019812039496895438 0.005251749153334179 0.0
0.005251749153334179 0.019812039496895438 0.03858507697191389
0.05022292778495918 0.04046812518812997 0.006956906412901003 -
0.025574203141729788 -0.022657586462979884 ];
WK1  = [0.513282 4.561110];
RK1 = [0.00011471274147674296 0.0053241944447029645
0.01375342775340076 0.02470037943468705 0.0366934031461834
0.04797609018238804 0.057015359357022385 0.06278230792236948
0.0647544760221452 0.06278230792236948 0.057015359357022385
0.04797609018238804 0.0366934031461834 0.02470037943468705
0.01375342775340076 0.0053241944447029645 0.00011471274147674296 ];
IK1 = [0.009115767931619433 0.013416076181066846 0.016518778005752112
0.01721496258662671 0.015063844631669552 0.010684250570423394
0.005570251097457814 0.0015288678619985174 0.0 0.0015288678619985174
0.005570251097457814 0.010684250570423394 0.015063844631669552
0.01721496258662671 0.016518778005752112 0.013416076181066846
0.009115767931619433 ];
RADIUS = 8;
%}

img = im2double(imread('test.jpg'));

tic
lol = convrgb(img,[1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1],'same');
```

```
toc

tic
gaussian_img = convrgb(img, fspecial('gaussian',[2*RADIUS+1
2*RADIUS+1], 2*RADIUS/3), 'same');
toc

tic
circle_img = convrgb(img, fspecial('disk', RADIUS), 'same');
toc

tic
final_img = convcomp(img, RK0, IK0, WK0) + convcomp(img, RK1, IK1,
WK1) + convcomp(img, RK2, IK2, WK2) + convcomp(img, RK3, IK3, WK3) +
convcomp(img, RK4, IK4, WK4);
toc


orig = figure;
imshow(img)

gauss = figure;
imshow(gaussian_img)
saveas(gauss,'gaussian.png');

circular = figure;
imshow(circle_img);
saveas(circular,'circular.png');

csc = figure;
imshow(final_img);
saveas(csc,'final.png');

diff = figure;
imshow((final_img - circle_img).^2);
saveas(diff,'diff.png');
```

Contents of convrgb.m:

```
function out = convrgb(img, kernelP, cfg)
kernel = gpuArray(kernelP);
redChannel = gpuArray(img(:, :, 1));
greenChannel = gpuArray(img(:, :, 2));
blueChannel = gpuArray(img(:, :, 3));

r = conv2(double(redChannel), kernel, cfg);
g = conv2(double(greenChannel), kernel, cfg);
b = conv2(double(blueChannel), kernel, cfg);

out(:,:,1) = r;
out(:,:,2) = g;
out(:,:,3) = b;
```

Contents of convcomp.m:

```
function out = convcomp(img, kernel_re, kernel_im, kernel_w)
kernel = kernel_re + 1j * kernel_im;
img_out = convrgb(img, kernel, 'same');
img_out = convrgb(img_out, transpose(kernel), 'same');
out = kernel_w(1) * real(img_out) + kernel_w(2) * imag(img_out);
```

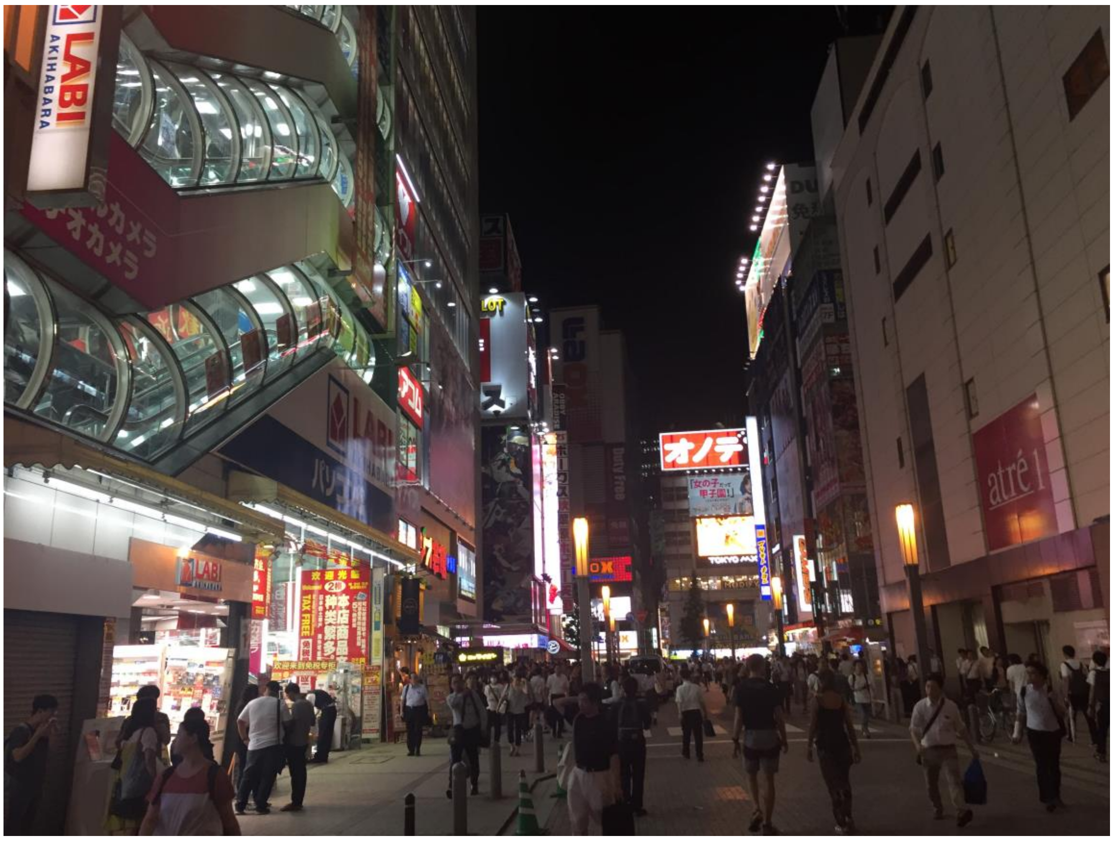

Figure 1: Original Image

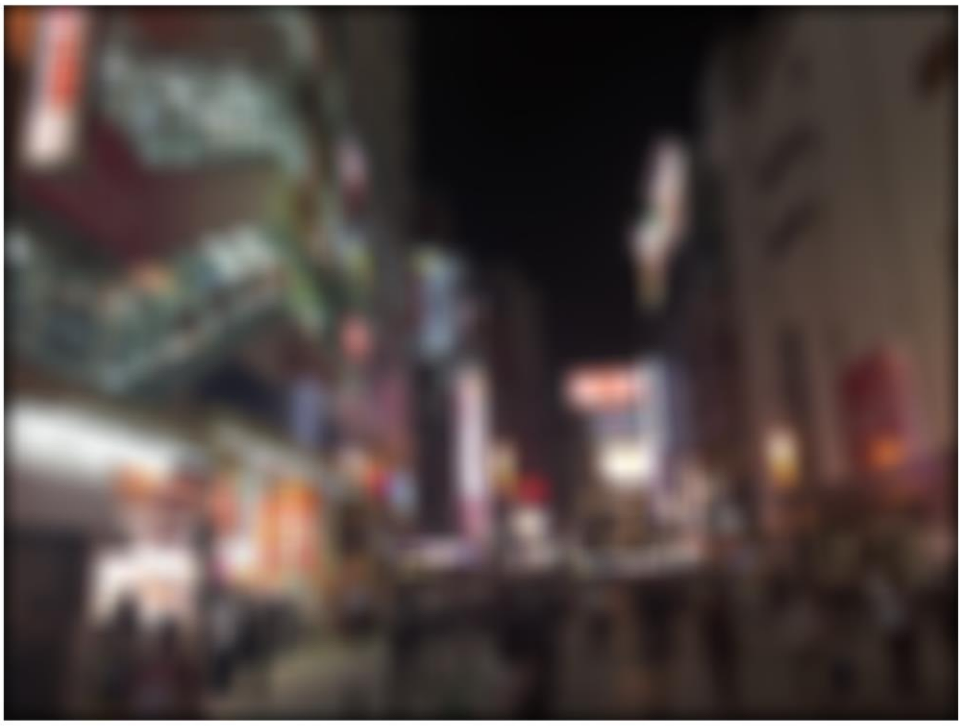

Figure 2: Gaussian Blurred Image

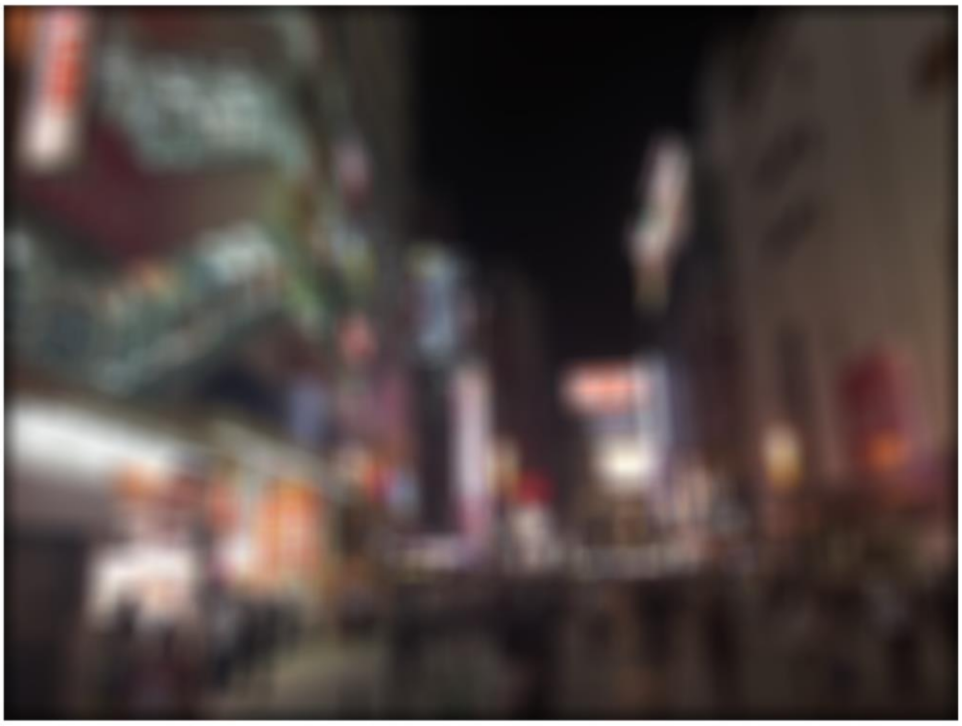

Figure 3: Circularly Blurred Image

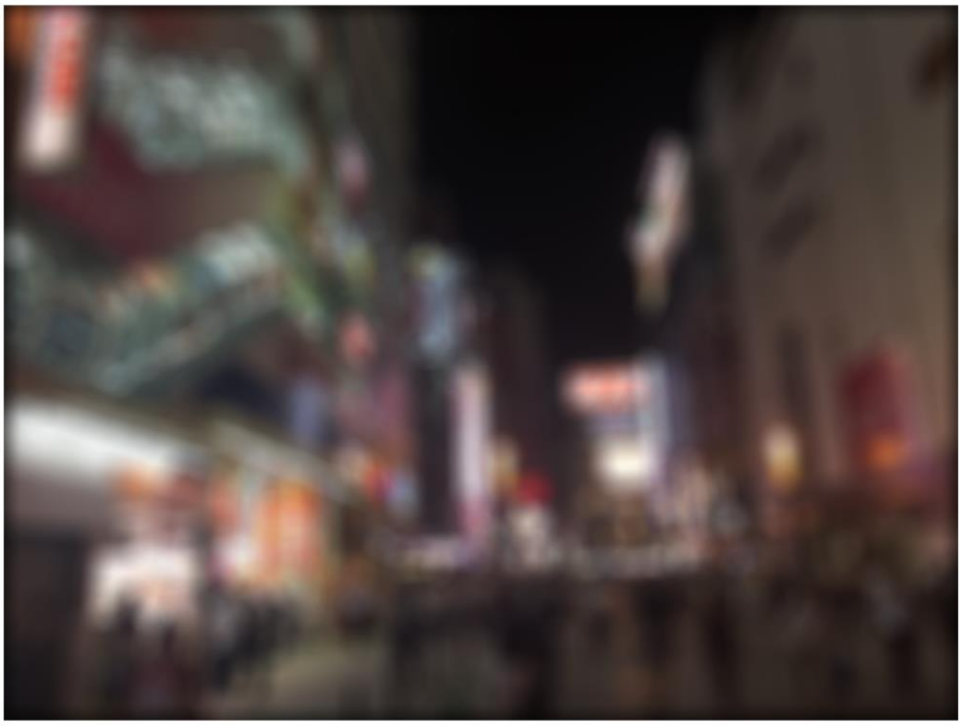

Figure 4: Circularly Separable Filter Output