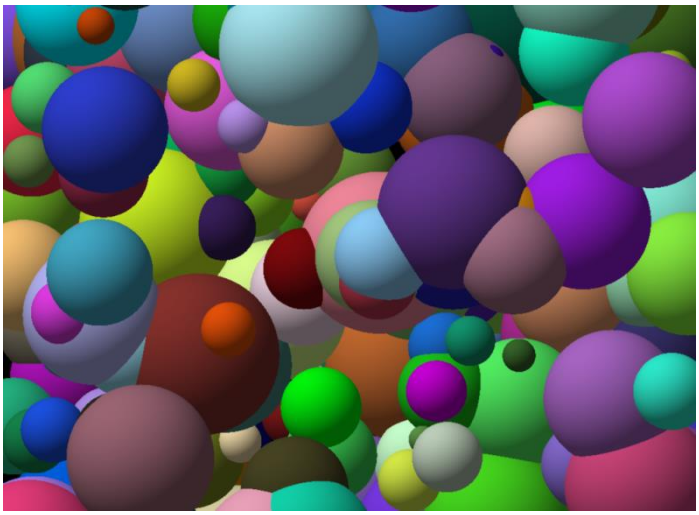
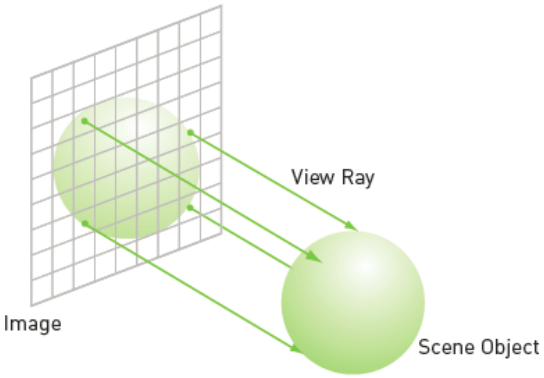
GPU Programming
TP2 – RayTracer



Ray tracing is a technique of generating an image of a 3D scene. Such images typically have a high degree of visual realism, usually higher than that of conventional rendering methods, but at a greater computational cost.

The approach is the following: We choose a spot in our scene to place an imaginary camera. The camera contains a light sensor, so to produce an image, we need to determine what light would hit that sensor. For that purpose, we imagine shooting a ray from each pixel and into the scene. In this way, each pixel behaves something like an eye that is “looking” into the scene:



Our ray tracer will be very simple. It will only support scenes of spheres, the camera is directed along the z-axis and restricted to moving within the xy-plane. The incidence of light is equally fixed to be parallel to the z-axis. The ray tracer will fire a ray from each pixel and keep track of which rays hit which spheres at which distance. In the case where a ray passes through multiple spheres, only the sphere closest to the camera can be seen.

We model our spheres with a data structure that stores its center coordinate (x, y, z), its radius, and its color (r, g, b).

```
#define INF 2e10f
struct Sphere {
	float r,g,b;
	float radius;
	float x,y,z;
	__host__ __device__ float hit(float cx, float cy, float *sh) {
		float dx  = cx - x;
		float dy  = cy - y;
		float dz2 = radius*radius - dx*dx - dy*dy;
		if (dz2>0) {
			float dz = sqrtf(dz2);
			*sh = dz / radius;
			return dz + z;
		}
		return -INF;
	}
};
```

The structure has a method called “hit”. Given a ray shot from the coordinates (ox, oy), this method determines whether the ray intersects the sphere. If the ray does intersect the sphere, it returns the distance from the camera where the ray hits the sphere, as well as the shading intensity (*sh ϵ[0…1]) of the pixel. You can add ambient light (a ϵ [0…1]) to the scene. In that case, the final shading intensity of a pixel is simply a+ (*sh) * (1-a).

Exercices (1 point per question)

You can reuse the base code provided for “TP1 – Julia sets” to get started.

1) Write a CPU version of the ray tracing algorithm. The user can change the number of spheres by hitting special keys and use the mouse to move the camera within the xy-plane.
2) GPU version 1: each thread is responsible for one pixel of the output image. Use pinned host memory for fast data transfers.
3) GPU version 2: load the sphere array into the constant memory of the GPU.
4) GPU version 3: use streams for task parallelization. 4 streams may be a good choice. Each stream processes a separate slice of the image.