PRAMOOK KHUNGUM: Hi, my name is Pramook Khungum. I'm supposed to be one of four, but it remains that I am the only one now. So my project was about global illumination. And it's about rendering 3D scenes.

So first, let me tell you about the direct illumination model. This is something you can achieve with ray tracing. As you can see, the image above is rendered by shooting a ray from your eyes, and then taking the local lighting properties of the materials, and then see whether there's a light coming to that point or not, and then figure out the color.

So in this model, there's a lot things unnatural about this. Because as we can see, the light is an area light, but then the shadow is very solid shadow as if it's lit by a point light or something like that. And you can see that the ceiling is not lit at all. This means that this model of lighting doesn't take into account the light that bounces between the walls and lighting parts that should have been lit in the real life scene, as you can see.

So global illumination basically tries to fix that. And the effects you can achieve with global illumination algorithms are basically soft shadows, color bleeding-- you can see that there's a very pale shades of blue here-- and caustics, which means that light from here goes through the glasses and then focus on here.

So at first when I started the project, I wanted to globally illuminate a very simple scene, which is the Cornell Box. So it's basically this box, and you replace these two spheres with two white boxes. And then you can observe color bleeding in something like that. And I wanted to do it in real time with caustics.

However, the things I accomplished are-- well, I globally illuminated the Cornell Box

as I promised. In real time-- no, it's too slow. With caustic, didn't implement photon mapping. And sadly, there's still a lot of room for optimization. So kind of disappointed myself a little bit, but yeah, that's fine.

So the algorithm I used for global illumination is called Instant Radiosity. It's by some German guy called Alexander Keller. It's good for scenes with diffuse objects, which means that scenes that doesn't have objects that reflect light or lets light transmit through. Diffuse objects can be just this table and something like that. Basically, it means that when light strikes it, it just bounces off randomly.

As such, it can't do caustics, although it is very simple to implement once you have the core. So let me describe the algorithm. You start with the light source, and then you emit artificial light particles and let them bounce around the scene. And then you will have some more light source. And then you ray trace using those light sources in the scene basically.

So this is the illustration of the algorithm. First, you have the light, which goes off from this square light. And then you illuminate the scene with it. And then you bounce some lights off, and then illuminate the scene. Then you combine the results, basically.

So in my implementation, at the beginning, I have one SPU scatter all the light particles. The number of light particles I scattered is about 128. And as they bounce around, it ends up to be about 250 light sources in total.

And then I rendered a scene by splitting the scene into horizontal slabs, and then just give the slabs to each SPU to render. In this way, since the scene is very small, it can just preload them to every SPU. And they all can work independently. So there's minimal communication going around.

And since I didn't implement any other smart data structure to take care of the geometry, it just used bruteforce ray tracing. My scenes have only eight objects anyway.

I wasn't maybe it might be a good point to see if there's [? a question. ?]

AUDIENCE: So the SPUs are doing rendering or ray tracing?

PRAMOOK KHUNGUM: Ray tracing, yes, basically.

AUDIENCE: But I mean, can a ray move from one slab to another depending on reflection or something?

PRAMOOK KHUNGUM: No. Basically, if you trace the light there, the SPU has all the ascription of the scene, so it can trace the ray throughout the scene. But it's responsible for only the pixels in the slab.

AUDIENCE: Oh, OK.

PRAMOOK KHUNGUM: The complexity is kind of bad because the time is you have your image size, the number of pixels times the number of light particles, which is about 200. And then since I used bruteforce ray shooting, you have to times by the size of objects.

But at first, I thought I have only eight objects anyway so it's not going to be a big performance rack. But then I came to the realization that this algorithm initially was conceived for using with programmable graphics hardware, which can do really dumb tasks like taking a point light source and illuminating a scene with shadows very fast.

So it turns out that this algorithm is not really suited for the [? Cell, ?] basically. I wish I could have done better on this. This 512 times 512 scene took 20 seconds to render on the [? Cell. ?] But the original paper, they rendered something a little bigger than this in 24 seconds with GPU basically, so it's kind of disappointing.

PROFESSOR SAMAN: [INAUDIBLE].

PRAMOOK KHUNGUM: Hello?

PROFESSOR SAMAN: 24 seconds on what type of a machine? Can you hear me?

PRAMOOK KHUNGUM: Yes, but not very clearly.

AUDIENCE: 24 seconds on what kind of a machine?

PROFESSOR SAMAN: The original paper took 24 seconds?

PRAMOOK KHUNGUM: 24 seconds on a 75 megahertz something machine with a GPU.

PROFESSOR SAMAN: OK. And did you try without using any SPU, just using CPU?

PRAMOOK KHUNGUM: Just using the CPU and the GPU and that one.

PROFESSOR: His question was, did you try using--

PROFESSOR SAMAN: If you just use CPU without any parallelism, how much time does it take?

PRAMOOK KHUNGUM: Oh, times that by 6. However, the speedup is proportional to the number of processors you used. I basically graphed the runtime with the number of processors. This is in [INAUDIBLE] scale. So as you can see, it-- so that's the best you can do about parallelism there, basically.

AUDIENCE: [INAUDIBLE].

PRAMOOK KHUNGUM: Yes?

AUDIENCE: So runtime is normalized to just the PPU?

PRAMOOK KHUNGUM: Using the SPU.

AUDIENCE: So what are the units for the y-axis?

PRAMOOK KHUNGUM: The y-axis is runtime in seconds.

AUDIENCE: I see. So 3.7 seconds, 3.6 seconds for one SPU?

PRAMOOK KHUNGUM: Yes. 3.6 seconds per one SPU. And as you increase, the times go down. You just divide that by that. Is that clear?

AUDIENCE: And they're all using the PPU as well?

PRAMOOK KHUNGUM: No, the PPU doesn't do anything. I just used--

AUDIENCE: Sorry, did you say 20 seconds on your previous slide?

PRAMOOK KHUNGUM: This is different [? image ?] slides.

AUDIENCE: 128 by 128.

PRAMOOK KHUNGUM: Yes.

AUDIENCE: Oh, OK. OK, different--

PRAMOOK KHUNGUM: Sorry, I didn't make that clear.

AUDIENCE: Sure, OK.

PRAMOOK KHUNGUM: This is 128. So it's about 16 times faster. I couldn't wait for llike--

AUDIENCE: Sure.

PRAMOOK KHUNGUM: Yeah. So here are some images I rendered. There are eight objects in here. And it took like 20.5 seconds with six SPUs. Another image that I rendered is a miniature of a Cornell Box. It took 20.2 seconds and six SPUs.

AUDIENCE: Why is the ceiling still so in black?

PRAMOOK KHUNGUM: Yes, I don't know that as well. Maybe it's about the distribution of the particle light source. Probably I think when it goes to the wall or to the floor, it gets severely attenuated that it cannot really light the ceiling basically.

So there's a comparison between my rendering and the real Cornell Box. Basically, yes, there are still a lot of [? boxes ?] around. Basically, the ceiling is not lit. And it's kind of strange that I see some gray tinges here instead of green as in this one. So probably, I have to fix that as well.

But as you can see, we have achieved soft shadows and then some color bleeding.

AUDIENCE: How do you determine the location of the light sources?

PRAMOOK KHUNGUM: How do I determine the--

AUDIENCE: Are you--

PROFESSOR: Are you giving this input to the algorithm, the location of the light source?

PRAMOOK KHUNGUM: Yes, basically.

AUDIENCE: And they can be arbitrary positions within the box or?

PRAMOOK KHUNGUM: Yes, basically. This one is just a square object, which is written in the scene description file. And I can locate them wherever I want basically. And the light particles goes down from here.

So there are some things that can be further optimized. Basically, if I had decided to use accelerated data structure, the complexity of that in that size times objects times the number of light sources, this term will be reduced to [? log. ?] And probably, you see about 3x speedup.

AUDIENCE: Can you give an example of an accelerated [? structure? ?]

PRAMOOK KHUNGUM: Basically, the most popular one is probably the-- what's it called? For example, [? the KD tree, ?] for example.

AUDIENCE: Oh, is it? [? OK. ?]

PRAMOOK KHUNGUM: And actually, I learned in class that basically, you can achieve about 2x or 3x speedup by rearranging the data so that you can trace four rays or 16 rays at a time instead of doing one at a time in the approach I'm taking. And if you do that, it's faster.

In the program, I have a lot of objects. And each of them has a transform associated to them so that I can locate them wherever I want and the object can be anything.

But there's another approach which says I just take anything to be a collection of triangles. And if you have triangles, the description is just three points in space and you don't need transform to do location or something like that. And basically, checking whether a ray hits a triangle or not will be faster.

So thank you very much.

PROFESSOR: A few more questions?

AUDIENCE: Do you know that work that was done here by Julie Dorsey in illumination?

PRAMOOK KHUNGUM: I have [? not. ?]

AUDIENCE: She basically used the trick of pre-computing, doing the ray tracing for some fixed

number of source locations and then just rendering, in real time, by taking a linear superposition of these ray-traced images. So long as you can approximate your light source by a linear combination of some fixed repertoire of light sources-- you didn't think about taking that shortcut presumably.

PRAMOOK KHUNGUM: I didn't think about that shortcut basically. I [INAUDIBLE] know something like that's--

PROFESSOR: [INAUDIBLE] few minutes left.

AUDIENCE: Good.

PRAMOOK KHUNGUM: Let's see. So [INAUDIBLE].

AUDIENCE: So is it possible to just connect it directly to a Playstation 3?

PRAMOOK KHUNGUM: So basically, there's this output.gga, which I'm going to remove because I'm going to generate it. The scene I'm going to render is going to be a Cornell Box with a cube and a sphere in it.

AUDIENCE: Pramook?

PRAMOOK KHUNGUM: Yes?

AUDIENCE: How easily could you change the light sources?

PRAMOOK KHUNGUM: I can just move the object and the light source.

AUDIENCE: OK, can you do that [INAUDIBLE]?

PRAMOOK KHUNGUM: Sure. Let's change it a point light source then. [INAUDIBLE] relax.

AUDIENCE: [INAUDIBLE].

[? PROFESSOR: It just ?] might be too slow.

PRAMOOK KHUNGUM: I see. Here we go. So right now, the current light source is this square. I'm going to comment it out and change it to the point light source here. And then here we go. That's too quick, but-- basically, there's a point light source located around here. So the scene is lit with one point light. So as to expected, the shadow is hard.

AUDIENCE: One more question.

PROFESSOR: One more question.

PRAMOOK KHUNGUM: Sure.

AUDIENCE: Just dividing the scene equally seems like it give good load balancing. If you had like roughly equal complexity across different sections-- if you have a really complex object or if you have reflections that go much deeper in some places in others, couldn't an equal division lead to bad load balancing? Did you think about other divisions?

PRAMOOK KHUNGUM: I suppose. But basically, other approach that I think of, which basically probably you can divide the scene in a really fine-grain manner.

AUDIENCE: Oh, you just do like [? that striping. ?]

PRAMOOK KHUNGUM: Yes.

AUDIENCE: Yeah, OK. So is there a reason that you chose [INAUDIBLE]?

PROFESSOR SAMAN: The communication-- how hard was it to get this communication working? So what [? you've ?] communicated, how often do you communicate?

PROFESSOR: Repeat the question.

PRAMOOK KHUNGUM: Professor Saman asked how often does the SPU communicate with one another, and the PPU, and how often do I do that?

PROFESSOR SAMAN: Yes.

PRAMOOK KHUNGUM: Basically, well, before the SPU starts running, the PPU sends it the scene description. And then it sends it the task that says, I want you to render this pixel to this pixel. And that's it. And then the SPU just streams the pixels to the main memory.

PROFESSOR SAMAN: OK, so you do a static balance, static distribution of work [? around? ?]

PRAMOOK KHUNGUM: Yes.

PROFESSOR SAMAN: And then when they're done, everybody sends it back, and then you synchronize?

PRAMOOK KHUNGUM: Yes.

PROFESSOR SAMAN: OK.

PRAMOOK KHUNGUM: Basically.

PROFESSOR: OK, let's thank the speaker.

[APPLAUSE]