# Raytracing with a twist

Nuit de l'Info 2021

Thibault Allançon

December 2021

# Introduction

# A little bit of context...

Towards the end of February 2021:

- ~~IMAGE~~GISTRE/RDI student

# A little bit of context...

Towards the end of February 2021:

- ~~IMAGE~~GISTRE/RDI student
- YAKA

# A little bit of context...

Towards the end of February 2021:

- ~~IMAGE~~GISTRE/RDI student
- YAKA
- A lot of free time!

# The raytracer project

**Definition**

*Ray tracing* is a technique used to render digital images.

# The raytracer project

## Definition

*Ray tracing* is a technique used to render digital images.

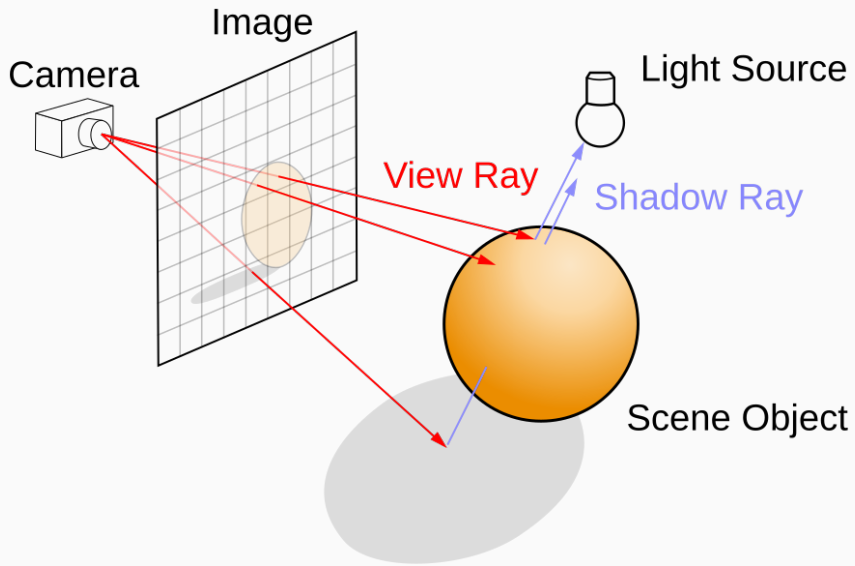



**Figure 1:** Basic principles

Source: https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

# The project specifications

- Students are free to code the ray tracer in any language
- Minimal set of features to implement before adding new ones
- At the end of the deadline: code submission + oral defense

# The project specifications

- Students are free to code the ray tracer in any language
- Minimal set of features to implement before adding new ones
- At the end of the deadline: code submission + oral defense

So... Where is the twist?

# Some inspiration

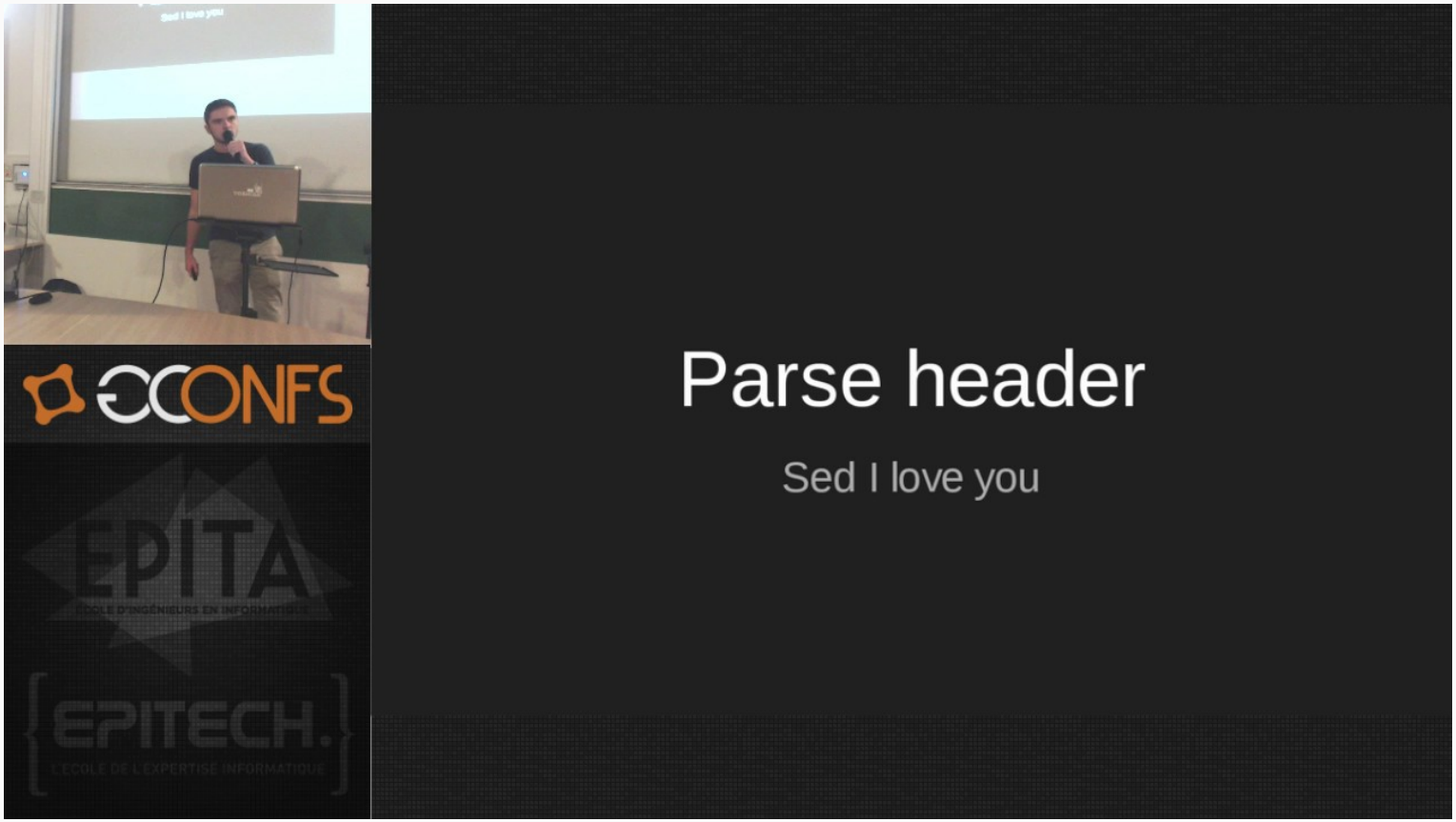



Figure 2: MyReadIso en Bash

# Some inspiration

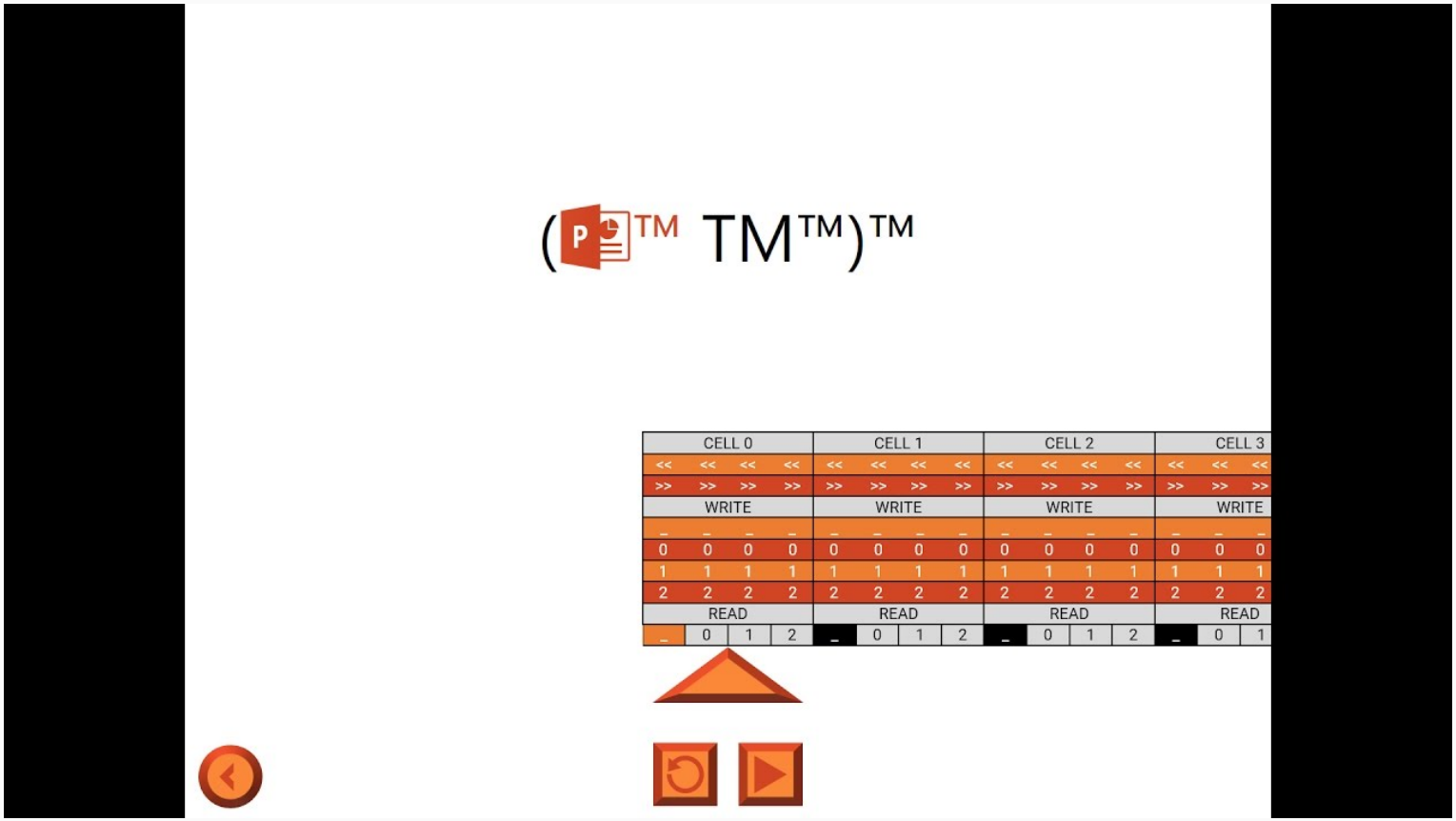



Figure 3: On The Turing Completeness of PowerPoint (SIGBOVIK)

# Some inspiration

Any **many** more examples:

- 93% of Paint Splatters are Valid Perl Programs
- WordTeX - A WYSIPCTWOTCG Typesetting Tool
- Compiling C to printable x86, to make an executable research paper
- 30 Weird Chess Algorithms: Elo World
- ...

# Which language should I pick?

Here are languages I have some experience with:

C, C++, Python, Java, SQL, OCaml, Rust, Bash, LaTeX

# Which language should I pick?

Here are languages I have some experience with:

~~C~~, ~~C++~~, ~~Python~~, ~~Java~~, SQL, OCaml, ~~Rust~~, Bash, LaTeX

# Which language should I pick?

Here are languages I have some experience with:

~~C~~, ~~C++~~, ~~Python~~, ~~Java~~, SQL, ~~OCaml~~, ~~Rust~~, ~~Bash~~, LaTeX

# Which language should I pick?

Here are languages I have some experience with:

~~C~~, ~~C++~~, ~~Python~~, ~~Java~~, ~~SQL~~, ~~OCaml~~, ~~Rust~~, ~~Bash~~, LaTeX

# What is LaTeX?

- LaTeX is a high-quality typesetting system
- Originally written by Leslie Lamport in the 1980s
- Free software based on TeX (Donald Knuth, 1970s)

# Your first LaTeX program

```
\documentclass{article}
\title{My awesome title}
\author{Xavier Login}
\date{January 2042}
\begin{document}
    \maketitle
    Hello world!
\end{document}
```

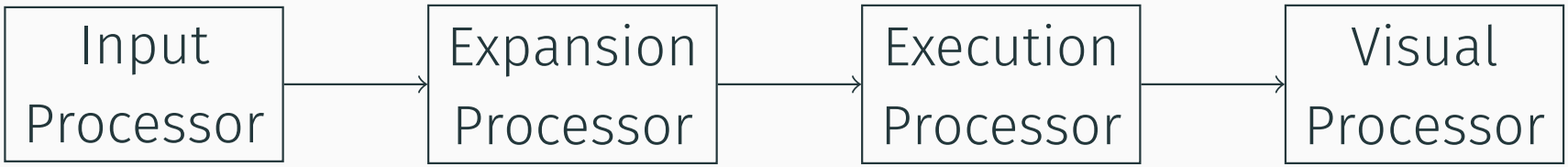
# Examples of crazy TEX programs

TEX is Turing complete!

- ICFP Contest 2008: Mars rover navigation
- Mandelbrot set
- A small C compiler

# TeX macros crash course

# TEX processor overview



# TEX macros 101

```
\def\mymacro{my replacement text}
Hello this is \mymacro!
```

# TEX macros 101

```
\def\mymacro{my replacement text}
Hello this is \mymacro!

\def\mymacro#1{my argument: #1}
Hello this is \mymacro{hi}!
```

# TEX macros 101

```
\def\mymacro{my replacement text}
Hello this is \mymacro!

\def\mymacro#1{my argument: #1}
Hello this is \mymacro{hi}!

\def\mymacro(#1,#2){(x=#1, y=#2)}
The coords are \mymacro(4,2)
```

```
\def\coords{(4,2)}
\def\mymacro(#1,#2){(x=#1, y=#2)}
The coords are \mymacro\coords
```

```
\def\coords{(4,2)}
\def\mymacro(#1,#2){(x=#1, y=#2)}
The coords are \mymacro\coords

% ! Use of \mymacro doesn't match its definition.
% l.3      The coords are \mymacro\coords
```

```
\def\coords{(4,2)}
\def\mymacro(#1,#2){(x=#1, y=#2)}
The coords are \mymacro\coords

% ! Use of \mymacro doesn't match its definition.
% l.3      The coords are \mymacro\coords
```

```
\def\coords{(4,2)}
\def\mymacro(#1,#2){(x=#1, y=#2)}
The coords are \expandafter\mymacro\coords
```

# TEX macros-ception

```
\a\b\c
% How to do the reverse expansion?
```

# TEX macros-ception

```
\a\b\c
% How to do the reverse expansion?

\expandafter\expandafter\expandafter\a\expandafter\b\c
```

# TeX macros expansion can be tricky

```
\def\mymacro#1{my arg: `#1'}
\def\tmp{(0,1,0)(0.5)}
\expandafter\mymacro\tmp
% Outputs: my arg: '('0,1,0)(0.5)
```

# TeX macros expansion can be tricky

```
\def\mymacro#1{my arg: `#1'}
\def\tmp{(0,1,0)(0.5)}
\expandafter\mymacro\tmp
% Outputs: my arg: '('0,1,0)(0.5)

\expandafter\mymacro\expandafter{\tmp}
% Outputs: my arg: '(0,1,0)(0.5)'
```

# Other useful commands

- `\edef` and `\noexpand`

# Other useful commands

- `\edef` and `\noexpand`
- `\meaning` and `\tracingmacros`

# It’s raytracing time

# Project outline

- Read as much as possible on raytracing and TEX macros

# Project outline

- Read as much as possible on raytracing and TEX macros
- User parameters to describe the world (spheres, lights, ...)

# Project outline

- Read as much as possible on raytracing and TEX macros
- User parameters to describe the world (spheres, lights, ...)
- Create a LATEX picture environment, one pixel at a time

# A first prototype

```
\documentclass{standalone}
\usepackage{color}
\usepackage{pgffor}
\usepackage{xfp}

\def\punit{1pt}
\def\pixel#1{\color[RGB]{#1}{\rule{\punit}{\punit}}}

\def\width{150}
\def\height{100}

\begin{document}
    \begin{picture}(\width,\height)
        \foreach \x in {0,...,\inteval{\width - 1}}{
            \foreach \y in {0,...,\inteval{\height - 1}}{
                \put(\x,\y){\pixel{255,0,0}}
            }
        }
    \end{picture}
\end{document}
```

# Step 0 - Vector data structure

We need a vector to represent coordinates and colors.

# Step 0 - Vector data structure

We need a vector to represent coordinates and colors.

```
% (x,y,z)
\def\getx(#1,#2,#3){#1}
\def\gety(#1,#2,#3){#2}
\def\getz(#1,#2,#3){#3}
```

# Step 0 - Vector data structure

We need a vector to represent coordinates and colors.

```
% (x,y,z)
\def\getx(#1,#2,#3){#1}
\def\gety(#1,#2,#3){#2}
\def\getz(#1,#2,#3){#3}

\def\vadd(#1)(#2){(%
    \fpeval{\getx(#1) + \getx(#2)},%
    \fpeval{\gety(#1) + \gety(#2)},%
    \fpeval{\getz(#1) + \getz(#2)}%
)}

\vadd(1,2,3)(4,5,6)
% Prints (5,7,9)
```

# Step 1 - Mr. Blue Sky

```
% #1 = ray (vect)
\def\createskypixel(#1){%
    \edef\unitdir{\vunit(#1)}%
    \edef\t{\fpeval{%
        0.5 * (\expandafter\gety\unitdir + 1.0)}%
    }%
    \edef\mt{\fpeval{1.0 - \t}}%
    \edef\va{\vmulscalar(1.0,1.0,1.0)\mt}%
    \edef\vb{\vmulscalar(0.5,0.7,1.0)\t}%
    \edef\sky{%
        \expandafter\expandafter\expandafter\vadd%
        \expandafter\va\vb%
    }%
    \expandafter\torgbpixel\sky%
}
```

# Step 1 - Result

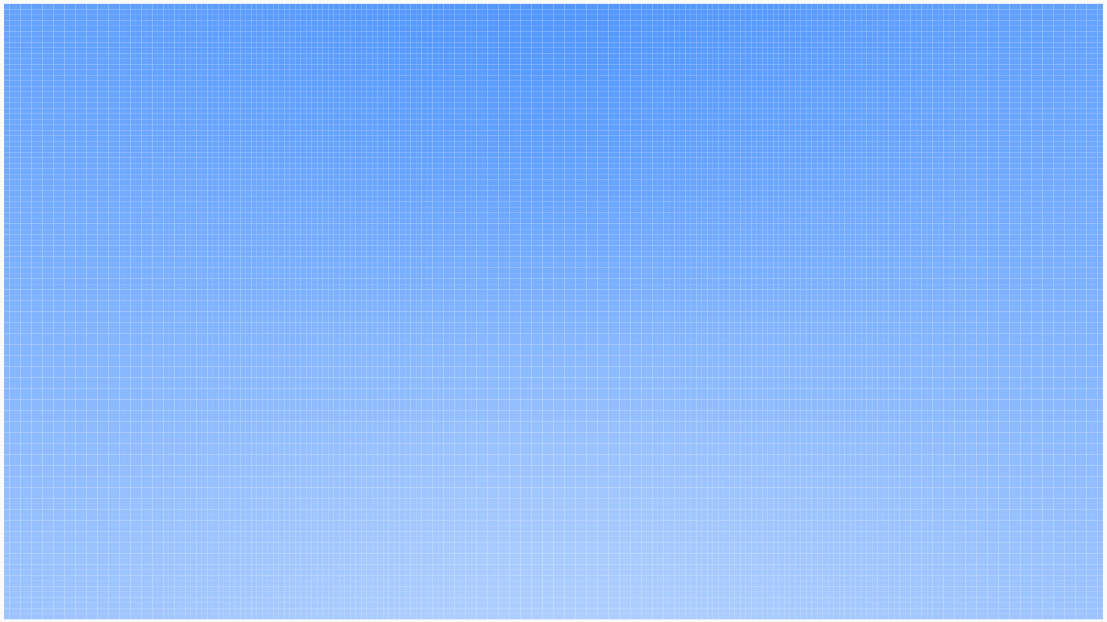


**Figure 4:** My first empty sky

# Step 2 - My first sphere!

```
% #1 = ray origin (point), #2 = ray dir (vect),
% #3 = sphere center (point), % #4 = sphere radius,
% #5 = macro continuation > 0, #6 = macro continuation <= 0
\def\hitspherediscriminant(#1)(#2)(#3)#4#5#6{%
    \edef\v{\fpeval{\vsub(#1)(#3)}}%
    \edef\a{\fpeval{\vdot(#2)(#2)}}%
    \edef\b{\fpeval{2.0 * \expandafter\vdot\v(#2)}}%
    \edef\vv{\v\v}%
    \edef\c{\fpeval{\expandafter\vdot\vv - #4*#4}}%
    \edef\discriminant{\fpeval{\b*\b - 4.0*\a*\c}}%
    \ifdim\dimexpr\discriminant pt>0pt #5\else #6\fi%
}
```

# Step 2 - My first sphere!

```
% #1 = ray origin (point), #2 = ray dir (vect),
% #3 = sphere center (point), % #4 = sphere radius,
% #5 = macro continuation > 0, #6 = macro continuation <= 0
\def\hitspherediscriminant(#1)(#2)(#3)#4#5#6{%
    \edef\v{\fpeval{\vsub(#1)(#3)}}%
    \edef\a{\fpeval{\vdot(#2)(#2)}}%
    \edef\b{\fpeval{2.0 * \expandafter\vdot\v(#2)}}%
    \edef\vv{\v\v}%
    \edef\c{\fpeval{\expandafter\vdot\vv - #4*#4}}%
    \edef\discriminant{\fpeval{\b*\b - 4.0*\a*\c}}%
    \ifdim\dimexpr\discriminant pt>0pt #5\else #6\fi%
}
% #1 = ray origin (point), #2 = ray dir (vect), #3 = macro cont.
\def\createpixel(#1)(#2)#3{%
    % [...]
    \edef\param{%
        \noexpand(#1)\noexpand(#2)\myspherepos{\mysphererad}%
        {\noexpand#3(1,0,0)}% Hit! (red pixel)
        {\noexpand#3\sky}% No hit (blue sky pixel)
    }%
    \expandafter\hitspherediscriminant\param%
}
```

# Step 2 - Result

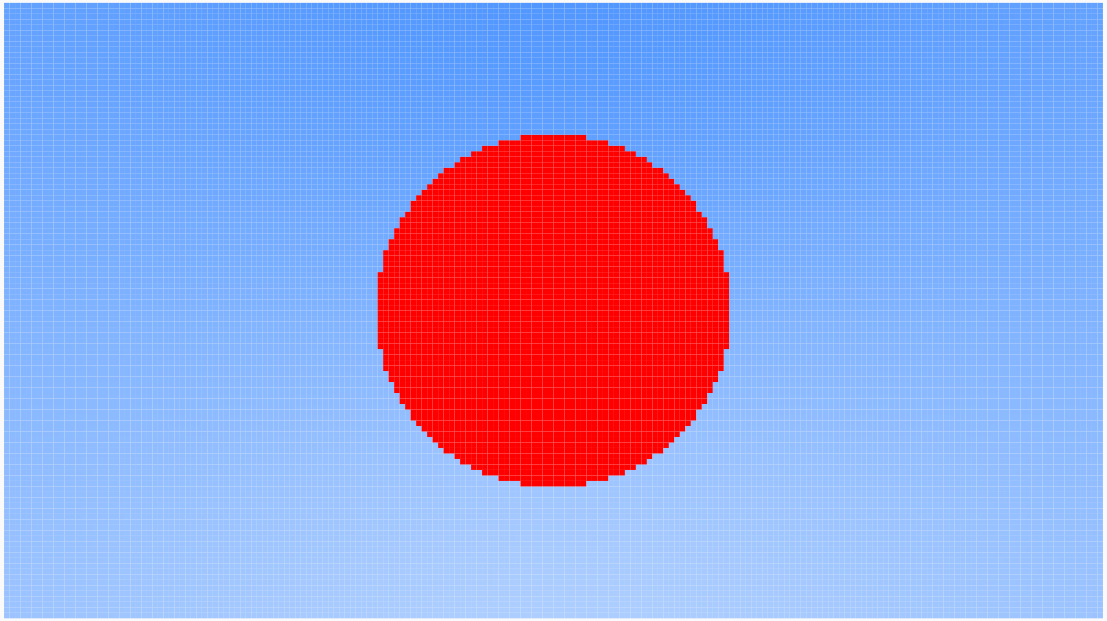


**Figure 5:** A nice red sphere

# Step 3 - Multiple spheres

```
% (#1) = (x,y,z), (#2) = radius, (#3) = (r,g,b) colors
\def\getspherecoords(#1)(#2)(#3){(#1)}
\def\getsphereradius(#1)(#2)(#3){#2}
\def\getspherecolors(#1)(#2)(#3){(#3)}
```

# Step 3 - Multiple spheres

```
% (#1) = (x,y,z), (#2) = radius, (#3) = (r,g,b) colors
\def\getspherecoords(#1)(#2)(#3){(#1)}
\def\getsphereradius(#1)(#2)(#3){#2}
\def\getspherecolors(#1)(#2)(#3){(#3)}

\def\world{%
    {(0,0,-1)(0.4)(1,0,0)},%
    {(-1,-0.5,-1)(0.2)(0,1,0)}%
}
```

# Step 3 - Multiple spheres

```
% Global variables to detect ray hit on sphere
\newcounter{hitsphere}
\def\hitspherecolor{(0,0,0)}
```

# Step 3 - Multiple spheres

```
% Global variables to detect ray hit on sphere
\newcounter{hitsphere}
\def\hitspherecolor{(0,0,0)}


% [...]

% #1 = ray origin (point), #2 = ray dir (vect)
\def\createpixel(#1)(#2){%
    \setcounter{hitsphere}{0}%
    \foreach \sphere in \world {%
        % call to \hitspherediscriminant [...]
    }%
    \ifnum\value{hitsphere}=1%
        \expandafter\createspherepixel\hitspherecolor%
    \else%
        % initialize parameters [...]
        \expandafter\createskypixel\ray%
    \fi%
}
```

# Step 3 - Result

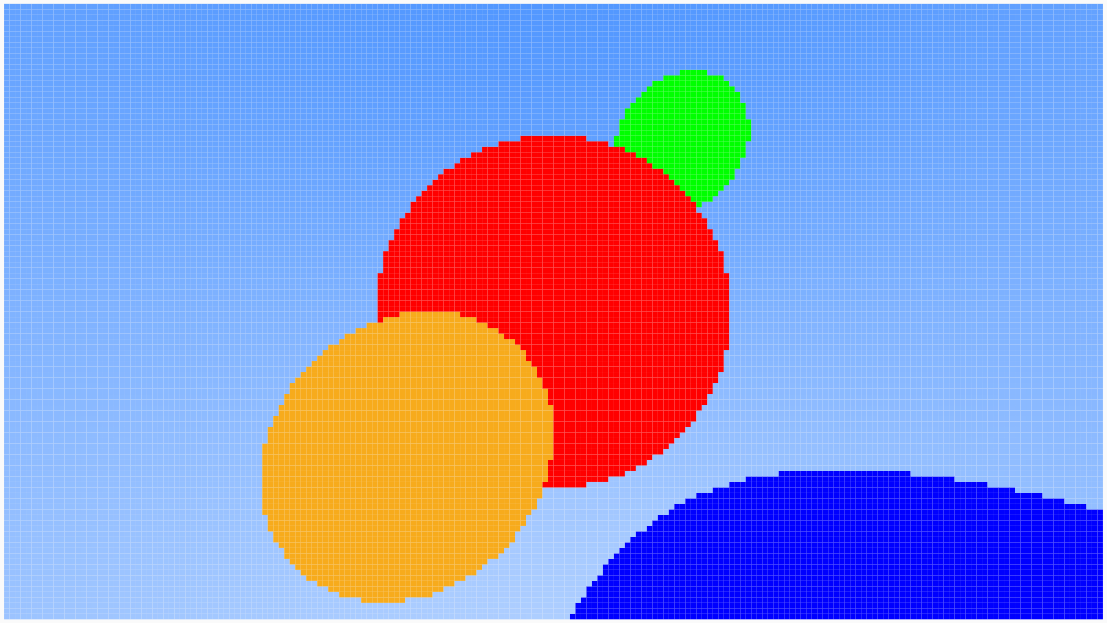


**Figure 6:** Many spheres

# Step 4 - Single light source

```
% (#1) = direction (vector), (#2) = color (r,g,b), (#3) = intensity
\def\getlightdir(#1)(#2)(#3){(#1)}
\def\getlightcolor(#1)(#2)(#3){(#2)}
\def\getlightintensity(#1)(#2)(#3){#3}
```

# Step 4 - Single light source

```
% (#1) = direction (vector), (#2) = color (r,g,b), (#3) = intensity
\def\getlightdir(#1)(#2)(#3){(#1)}
\def\getlightcolor(#1)(#2)(#3){(#2)}
\def\getlightintensity(#1)(#2)(#3){#3}
```

More linear algebra using TEX macros.

- https://en.wikipedia.org/wiki/Lambertian_reflectance
- https://raytracing.github.io/books/RayTracingInOneWeekend.html#diffusematerials
- https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/diffuse-lambertian-shading
- ...

# Step 4 - Result

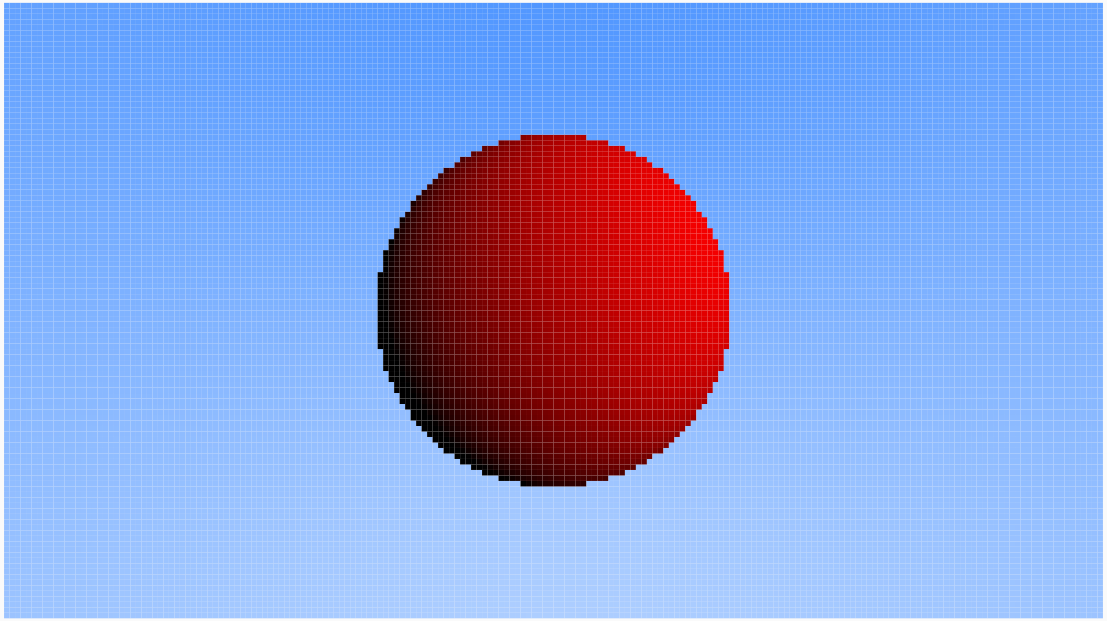


**Figure 7:** Directional light on sphere (example 1)

# Step 4 - Result

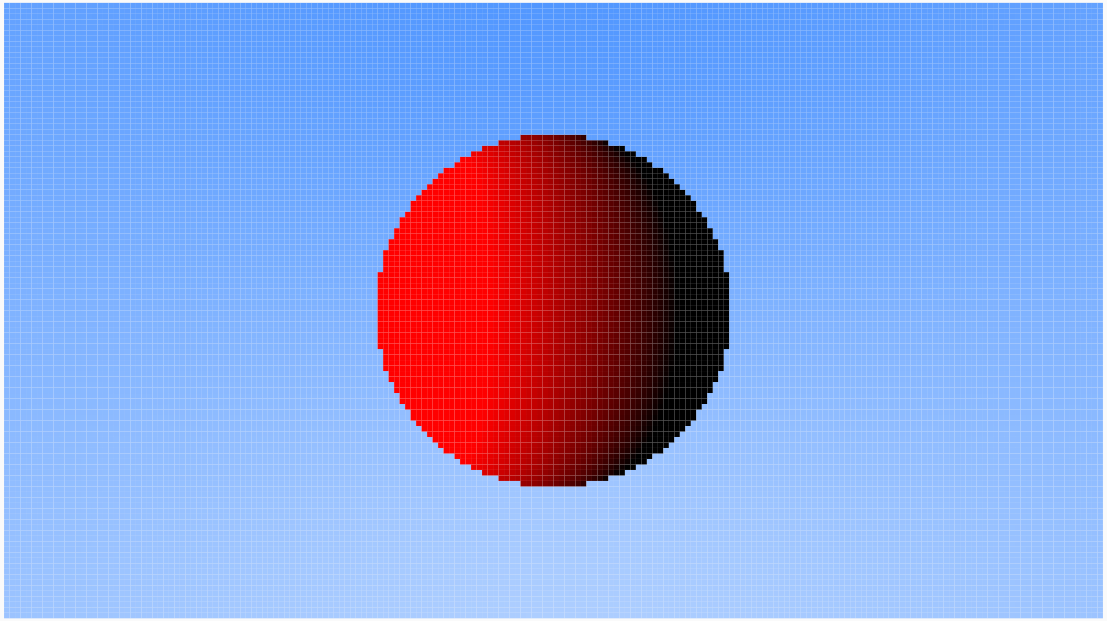


**Figure 7:** Directional light on sphere (example 2)

# Step 5 - Material component

For each sphere, let us add a material component:

- *raw*
- *lambertian*

# Step 5 - Material component

For each sphere, let us add a material component:

- *raw*
- *lambertian*

```
% (#1) = (x,y,z), (#2) = radius, (#3) = material name, (#4) = material args
\def\getspherecoords(#1)(#2)(#3)(#4){(#1)}
\def\getsphereradius(#1)(#2)(#3)(#4){#2}
\def\getspherematerialname(#1)(#2)(#3)(#4){#3}
\def\getspherematerialargs(#1)(#2)(#3)(#4){#4}

\def\world{%
    {(0,0,-1)(0.4)(raw)({(1,0,0)})},%
    {(-1,-0.5,-1)(0.2)(lambertian)({(0,1,0)(1.0)})}%
}
```

# Step 5 - Material component

For each sphere, let us add a material component:

- *raw*
- *lambertian*

```
% (#1) = (x,y,z), (#2) = radius, (#3) = material name, (#4) = material args
\def\getspherecoords(#1)(#2)(#3)(#4){(#1)}
\def\getsphereradius(#1)(#2)(#3)(#4){#2}
\def\getspherematerialname(#1)(#2)(#3)(#4){#3}
\def\getspherematerialargs(#1)(#2)(#3)(#4){#4}

\def\world{%
    {(0,0,-1)(0.4)(raw)({(1,0,0)})},%
    {(-1,-0.5,-1)(0.2)(lambertian)({(0,1,0)(1.0)})}%
}
```

How to implement multiple materials properly?

# Step 5 - Dynamic dispatch

```
% (#1) = intersection point (sphere hit by ray)
\def\createspherepixel(#1){%
    \edef\matname{\expandafter\getspherematerialname\hitsphere}%
    \edef\matargs{\expandafter\getspherematerialargs\hitsphere}%
    \def\fn{\csname createspherepixel\matname\endcsname}%
    \edef\args{(#1)\matargs}%
    \expandafter\fn\args%
}
```

# Step 5 - Dynamic dispatch

```
% (#1) = intersection point (sphere hit by ray)
\def\createspherepixel(#1){%
    \edef\matname{\expandafter\getspherematerialname\hitsphere}%
    \edef\matargs{\expandafter\getspherematerialargs\hitsphere}%
    \def\fn{\csname createspherepixel\matname\endcsname}%
    \edef\args{(#1)\matargs}%
    \expandafter\fn\args%
}

\def\createspherepixelraw(#1)(#2)
    % [...]
\def\createspherepixellambertian(#1)(#2)(#3)
    % [...]
```

# Final rendering

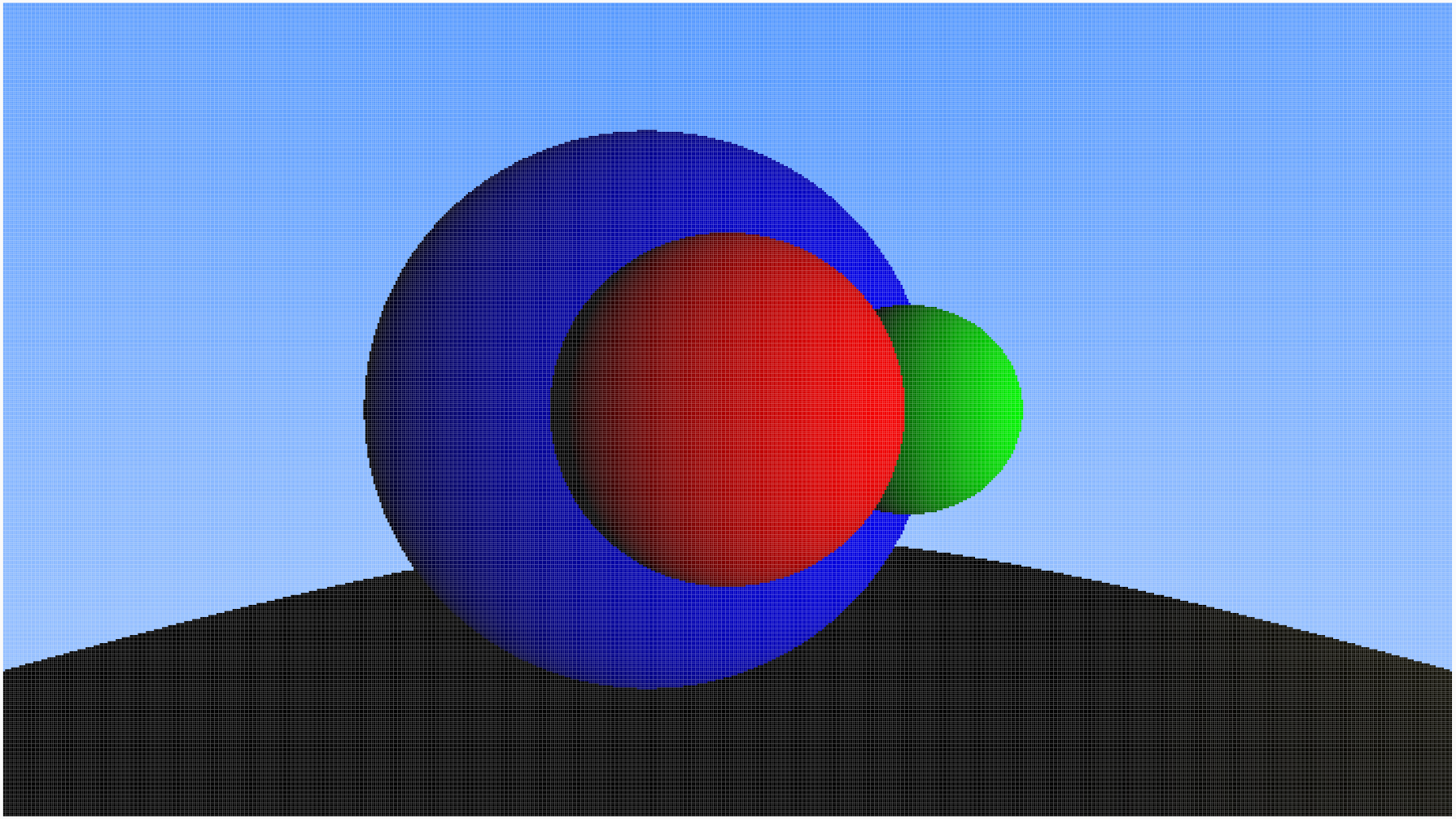


**Figure 8:** It takes ~1h20 to render 720p images

# Conclusion

# Going further

Missing features:

- Multiple light sources
- Sphere reflection
- Cast shadows from other spheres
- Anti-aliasing
- ...

# Wrapping up

- Embrace stupid side project ideas

# Wrapping up

- Embrace stupid side project ideas
- Challenge yourself and explore new ways of doing something

# Wrapping up

- Embrace stupid side project ideas
- Challenge yourself and explore new ways of doing something
- Learn a bit of LaTeX :)

# One more thing...

# One more thing...

```
\usepackage{raytracer}
```

# One more thing...

```
\usepackage{raytracer}
```

**Fun fact**

All the examples in the slides are generated by the raytracer.
It took **~4 hours** to compile the slides!

# Resources

- Source code: https://github.com/haltode/raytracer
- Raytracing resources
  - https://raytracing.github.io/
  - https://www.pbr-book.org/
- LaTeX resources
  - Books: *The TeXbook*, *TeX by Topic*, *The LaTeX companion*, ...
  - http://pgfplots.sourceforge.net/TeX-programming-notes.pdf
  - https://www.tug.org/TUGboat/tb09-1/tb20bechtolsheim.pdf
  - https://www.overleaf.com/learn/latex/Articles
  - https://zestedesavoir.com/tutoriels/826/introduction-a-latex/
  - Another TeX raytracer! :D
    https://github.com/hak7a3/raytracing-tex

# Questions?

Thanks for listening!

Thibault Allançon
thibault.allancon@epita.fr
haltode @ irc.libera.chat