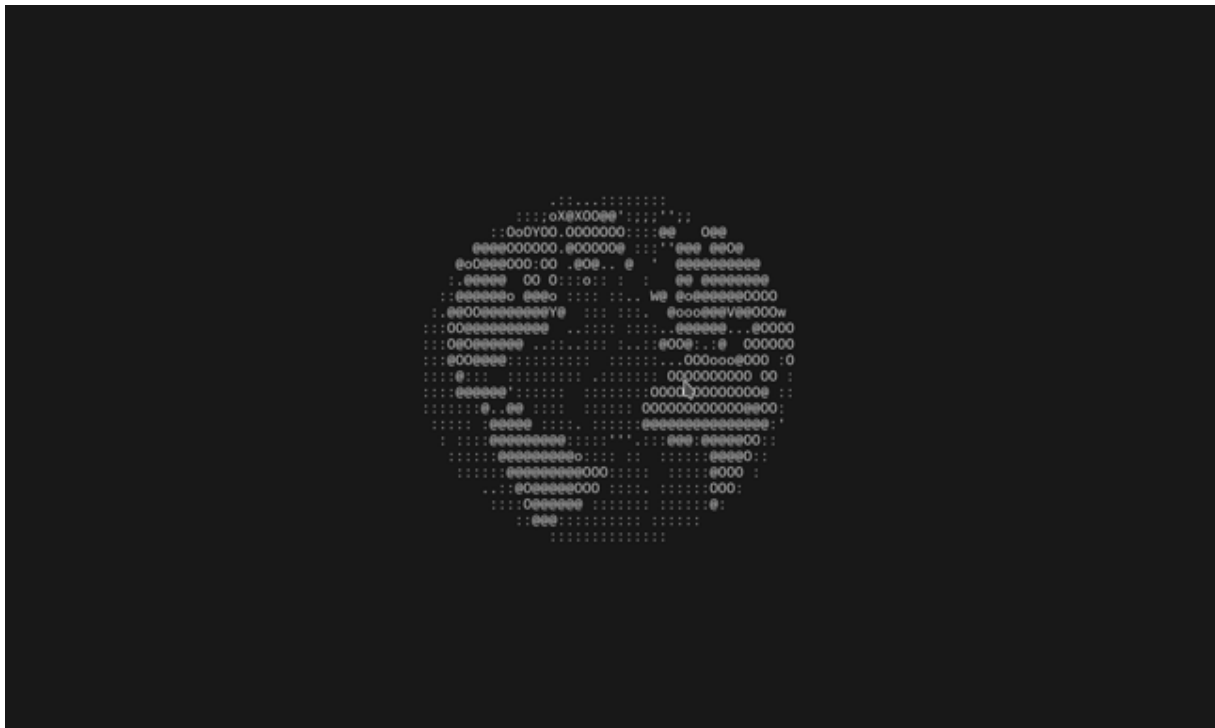




Render an ASCII globe in your terminal. Make it interactive or just let it spin in the background.



Changelog

v0.2.1: - upgraded `clap` dependency to 3.0.0 - changed `globe-cli template` argument to not be required

v0.2.0: - added multiple CLI arguments for setting up the scene (`refresh-rate`, `globe-rotation`, `cam-rotation`, `cam-zoom`, `location`, `focus-speed`, `night`, `template`, `texture`, `texture-night`) - added experimental `listing mode` that supports reading coordinates from standard input and going through all of them, animating camera target changes (see `--pipe`) - enabled ability to display night side of the globe using an additional texture - changed default Earth texture (now includes New Zealand) - added vim-style navigation for the interactive mode - improved

internal library representation of [Texture](#) - improved documentation

v0.1.2 - added clearing screen on exit - fixed panic when using rust version <1.45

v0.1.1 - fixed mouse capture staying on after exit

v0.1.0 - initial release

Install

To build [globe-cli](#) you will need to have Rust programming language installed on your machine.

Use [cargo install](#):

```
1 cargo install globe-cli
```

Or [git clone](#) and [cargo run --release](#) directly from the repository.

AUR

[globe](#) can be installed from available AUR packages using an AUR helper. For example,

```
1 yay -S globe-cli
```

If you prefer, you can clone the AUR packages and then compile them with makepkg. For example,

```
1 git clone https://aur.archlinux.org/globe-cli.git
2 cd globe-cli
3 makepkg -si
```

Docker

You can also use Docker to try out [globe](#), no Rust needed. After cloning the repo, just build and run an image from the [Dockerfile](#) contained at the root of the project:

```
1 docker build -t globe .
2 docker run -it --rm globe -s
```

Run

To get a full listing of available features and options, show the [--help](#) information with:

```
1 globe -h
```

Display a globe in *screensaver mode* using the `-s` option.

```
1 globe -s
```

It's kind of boring. Let's add some camera rotation to make it look more alive:

```
1 globe -sc2
```

Now let's also enable the night side and rotate the globe on its axis:

```
1 globe -snc2 -g10
```

If you want to adjust things at runtime check out the *interactive mode*. Here you can pan the globe around using either the mouse or keyboard arrows:

```
1 globe -i
```

Use `+` and `-` to control the globe rotation speed, `,` and `.` to control the camera rotation speed, `PgUp` and `PgDown` to control the camera zoom, `n` to toggle displaying globe's night side.

Settings we used on the *screensaver mode* also work:

```
1 globe -inc2 -g10
```

Last but not least there is the *listing mode*. It allows you to pass location coordinates to the program and see them shown one by one on the globe. Currently, it only supports a very basic input format. Here's an example:

```
1 echo "0,0.5;0.1,0.5;0.3,0.5;0.5,0.5;0.7,0.5" | globe -p
```

If you're feeling creative, you can also load custom textures, like so:

```
1 globe -in --texture ./path-to-texture --texture-night ./path-to-night-  
  texture
```

Use the library

To use `globe` within your Rust project, add it to your dependencies:

```
1 [dependencies]  
2 globe = "0.2.0"
```

First create a `Globe`:

```
1 let mut globe = GlobeConfig::new()  
2   .use_template(GlobeTemplate::Earth)  
3   .with_camera(CameraConfig::default())
```

```
4     .build();
```

Next make a new `Canvas` and render the `Globe` onto it:

```
1 let mut canvas = Canvas::new(250, 250, None);
2 globe.render_on(&mut canvas);
```

You can now print out the canvas to the terminal:

```
1 let (size_x, size_y) = canvas.get_size();
2 // default character size is 4 by 8
3 for i in 0..size_y / 8 {
4     for j in 0..size_x / 4 {
5         print!("{}", canvas.matrix[i][j]);
6     }
7     println!();
8 }
```

See `globe-cli` code for examples of runtime changes to the `Globe` and it's `Camera`.

Credits

Rendering math based on C++ code by DinoZ1729.