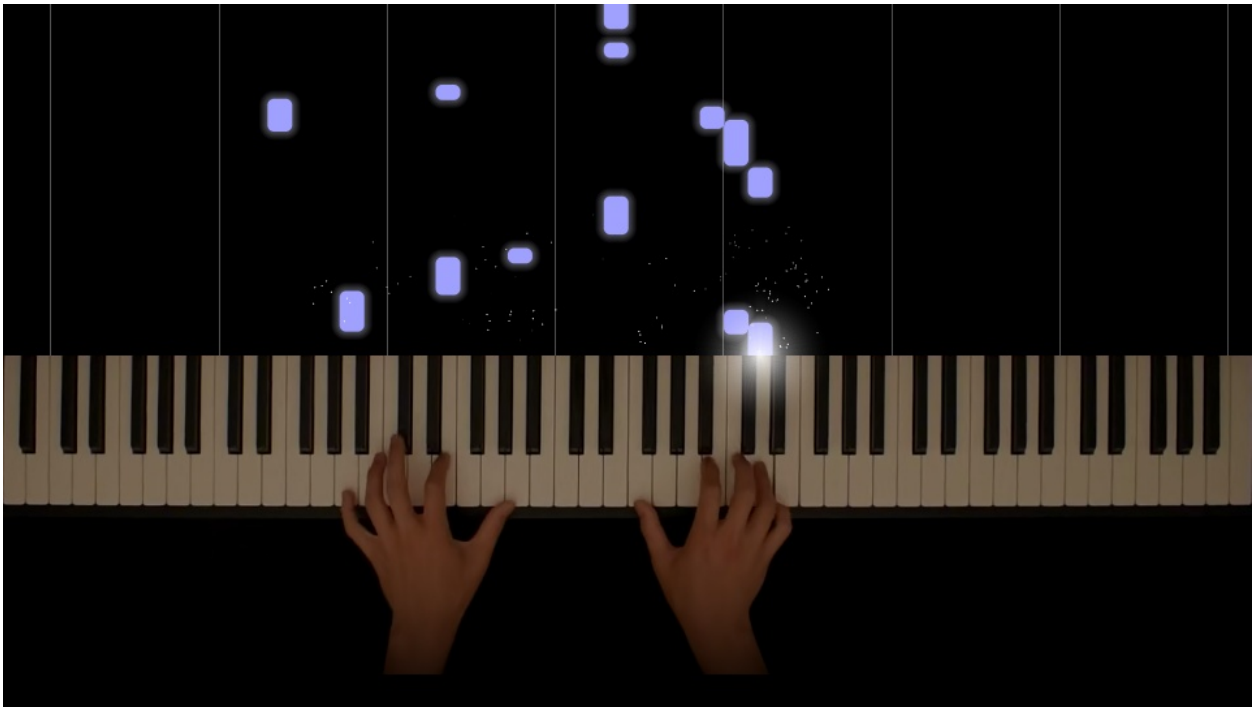
PianoRay

Release 0.2.4

PianoRay Authors

Jun 06, 2022

GENERAL

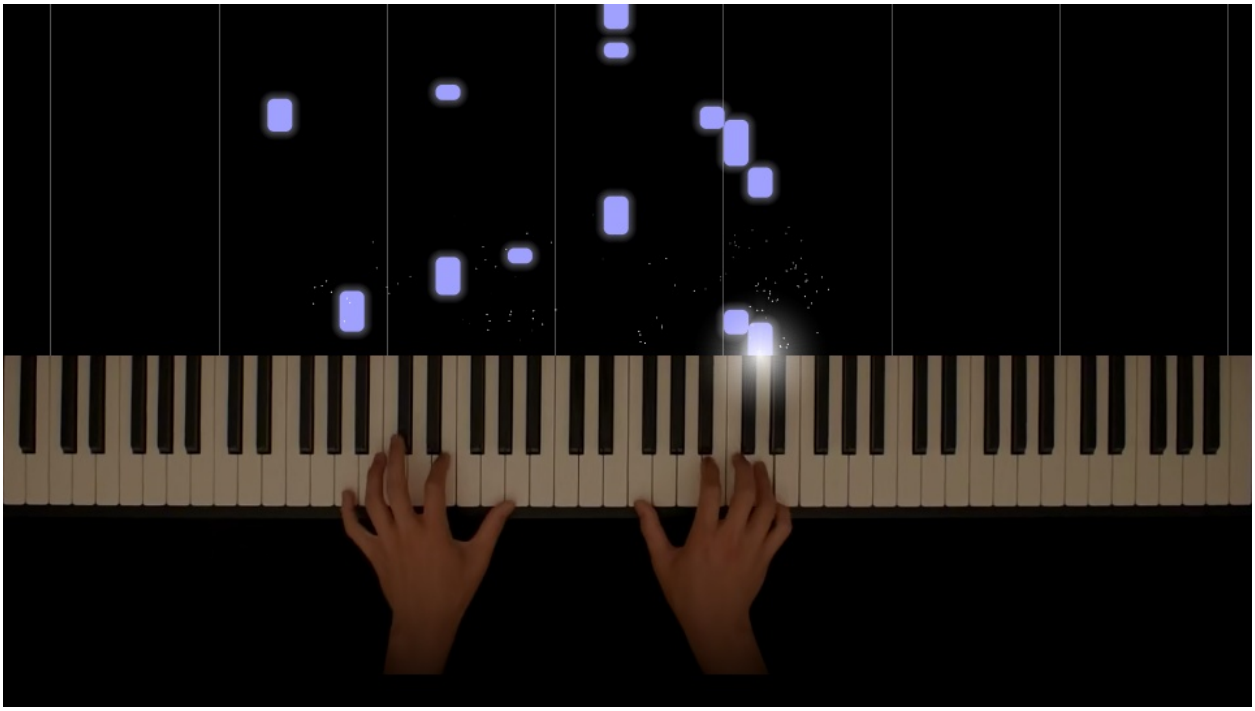


CHAPTER

ONE

ABOUT

PianoRay is a piano visualization tool.



1.1 Features

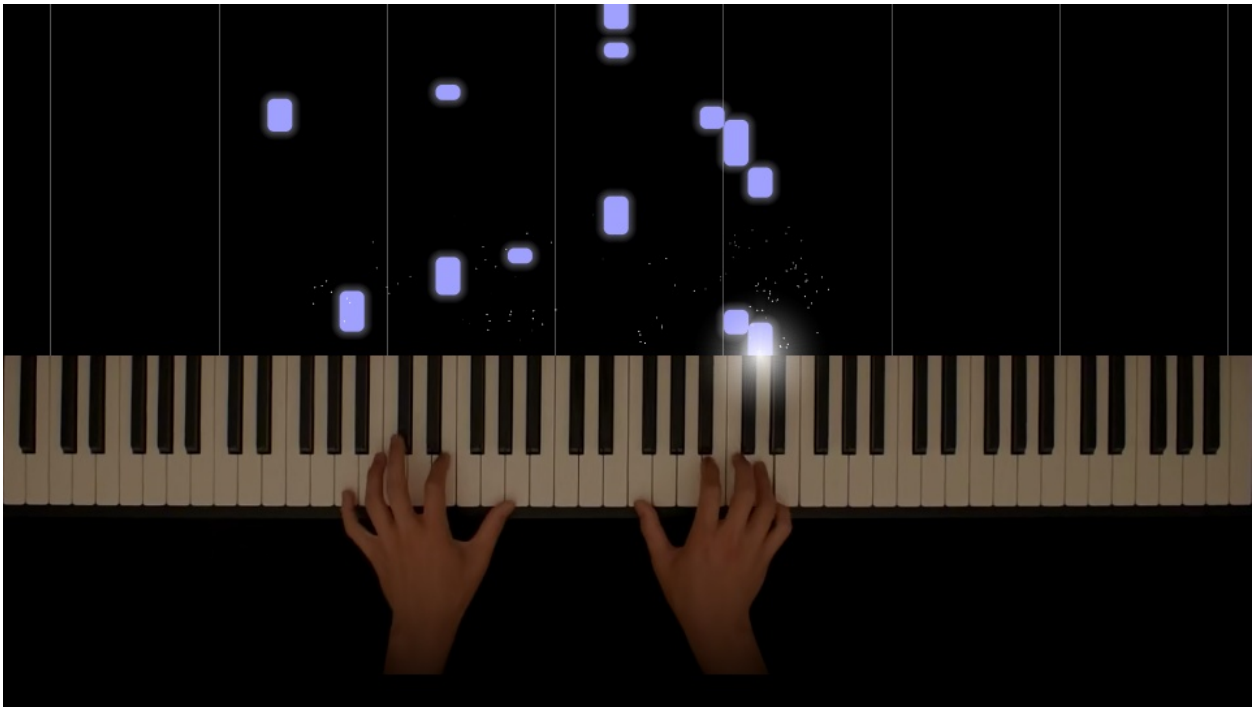
- Adds dropping blocks.
- Crops piano from video.
- Glare when blocks hit piano.
- Automatically compiles audio and video.

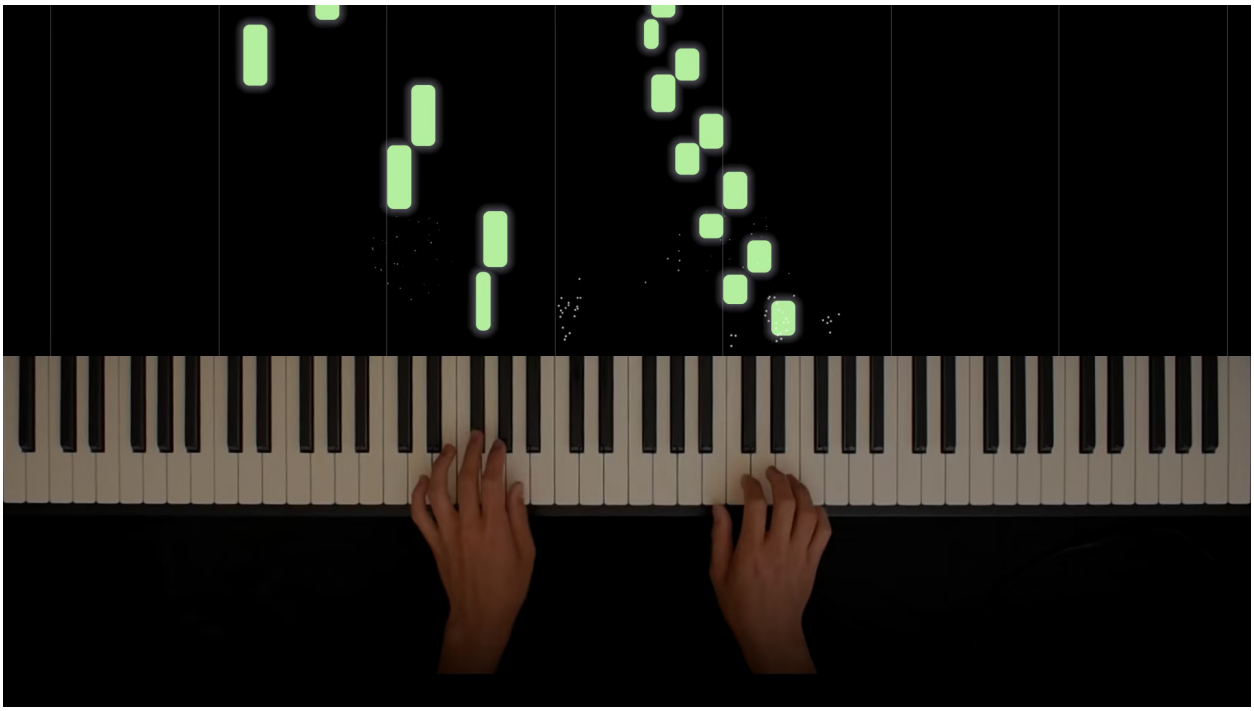
CHAPTER

TWO

GALLERY

Example renders.





CHAPTER

THREE

INSTALLATION

3.1 Dependencies

- Python version 3.8 or higher.
- FFmpeg.
- C++ compiler (`g++`).
- Python packages listed in `requirements.txt`
- Basic Python knowledge.

3.2 Latest

PianoRay is distributed on PyPI. Install with

```
pip install pianoray
```

3.3 Master Branch

May be unstable or have bugs.

```
pip install git+https://github.com/phuang1024/pianoray
```

CHAPTER

FOUR

SUPPORT

Please open an issue or discussion on GitHub.

CHAPTER

FIVE

LICENSE

PianoRay is licensed under GNU GPL v3. See `LICENSE` for the full license text.

You are free to use the software however you want.

If you are using the code itself, i.e. using some code from PianoRay for your own projects, you must license the complete derived work with a compatible license.

If you are just using the output i.e. the rendered videos, you may use the output however you want, with or without credit.

CHAPTER

SIX

FIRST VIDEO

First, install PianoRay. Follow instructions in General/Installation.

6.1 Example Files

Download example performance files. This script copies the Fur Elise example recording to `~/pianoray_tutorial`.

```
cd /tmp
git clone https://github.com/phuang1024/pianoray
cd pianoray/examples/furelise

mkdir ~/pianoray_tutorial
cp video.mp4 midi.mid audio.mp3 ~/pianoray_tutorial
```

The video file contains the recording of the piano. The MIDI file contains data about which notes are played. The audio file has the audio.

6.2 Create Settings

In order to convey settings to PianoRay, we use the Python API. The API usage is described in detail in *Animation*.

Save this data to `~/pianoray_tutorial/furelise.py`:

```
from pianoray import *

class FurElise(DefaultScene):
    def setup(self):
        self.video.resolution = (960, 540)
        self.video.fps = 30

        self.midi.file = "midi.mid"

        self.audio.file = "audio.mp3"
        self.audio.start = 20.74

        self.keyboard.file = "video.mp4"
        self.keyboard.start = 4.75
        self.keyboard.end = 37.64
        self.keyboard.crop = ((252,480), (1793,487), (1789,676), (257,666))
```

This creates a new scene called `FurElise` with some settings. PianoRay will read the scene to obtain settings.

6.3 Render

To start the render, run these commands in a shell:

```
cd ~/pianoray_tutorial
pianoray render furelise.py FurElise -o out.mp4 -p
```

This starts rendering, using the provided Python script and class name. The `-p` flag tells PianoRay to open the output file after rendering.

Rendering may take a few minutes. If the renderer crashes, run the same command again. If it repeatedly does not work, open an issue on GitHub for help.

CHAPTER

SEVEN

SECOND VIDEO

In this section, we will explore the API further.

Please follow the steps in *First Video* first, as we will use the setup and code as a starting point.

7.1 Basics

The `Scene` class extends `DefaultScene`, which is defined internally. This scene contains *`PropertyGroup`* instances, each with their own *`Property`*.

When we run code like `self.video.resolution = ...`, we are setting the value of the property. The available properties are documented in *Properties*. The current properties we have changed are either not exciting or mandatory.

Let's change the appearance of the video.

7.2 Changing Props

There is an available property `blocks.color`, which allows us to set the RGB color of the blocks.

Add this line somewhere in the `setup` method:

```
self.blocks.color = (255, 160, 160)  # Red
```

This will set the blocks to red. Run the render command to re-render the video:

```
pianoray render furelise.py FurElise -o out.mp4 -p
```

You may be asked whether you want to overwrite the output file. Choose yes. The new video will have red blocks.

7.3 Animation

In addition to setting values, the API also supports animating them. This is done with keyframes.

Essentially, a keyframe contains a frame, a value, and an interpolation. The frame is which frame the keyframe is. The value is the value at the frame. The interpolation is how to transition from this keyframe to the next. This is described in detail in *Animation*.

Warning: If a property is animated, PianoRay will ignore it's non-animated value:

```
# This value will be used if there is no animation.
self.blocks.color = ...

# The animation will be used, but the value from the previous line
# will be ignored.
self.blocks.color.animate(...)
```

Animation is done with a property's `animate` method. Let's animate the blocks changing from green to blue. Add these lines somewhere:

```
self.blocks.color.animate(
    (100, (160, 255, 160), Interp.LINEAR),  # Green
    (150, (160, 160, 255), Interp.LINEAR),  # Blue
 )
```

Render the video again and you should see the blocks change color somewhere in the middle. Notice how the blocks are not red, even though the line above sets them to red. Read the warning above to learn why.

CHAPTER

EIGHT

ANIMATION

Animation is done on *pianoray.Property* with `Keyframe` instances. Each keyframe contains a frame, a value, and an interpolation.

8.1 Syntax

Call the `animate` method on *pianoray.Property*:

```
def setup(self):
    # All syntaxes do the same thing.
    self.group.prop.animate(frame, value, interp)
    self.group.prop.animate((frame, value, interp))
    self.group.prop.animate(
        (frame, value, interp),
        (frame2, value2, interp2),
    )
    self.group.prop.animate(Keyframe(frame, value, interp))
```

CHAPTER

NINE

RECORDING

Instructions for recording and making your own video.

9.1 Recording

You will need to record two files: Video and MIDI. In order to record these files, you will need a MIDI keyboard, a camera, and a computer.

9.1.1 Video

Find a setup with a camera looking down vertically onto the keyboard. Some things to consider:

- Safety: Make sure the camera won't fall down.
- Stability: Try to reduce shaking, e.g. from vibrations from the keyboard.
- Focus: Make sure you focus the camera onto the keyboard before recording. It is very disappointing to find that the video is ruined because the keyboard recording is blurry (speaking from experience).
- Background: If you desire, place a dark tarp under the keyboard so you can create the "hands floating over nothing" effect. There are some settings in PianoRay to dim the background and achieve this effect.
- Privacy: If you plan to release the video to the public, make sure it doesn't contain any private information.

9.1.2 MIDI

Connect the MIDI keyboard to the computer. Use MIDI recording software to record the MIDI. I use MidiEditor, which has worked great.

9.2 Processing

9.2.1 Audio

Create an audio file from the MIDI.

1. Download a soundfont. SoundFonts4U has great piano soundfonts.
2. Install software that can render a MIDI file. I use FluidSynth, and the rest of these instructions assume you have FluidSynth.

3. Run this command, which uses FluidSynth to render and FFmpeg to write the audio file: `fluidsynth -a alsa -T raw -g GAIN -F - SOUNDFONT.sf2 MIDI.mid | ffmpeg -y -f s32le -i - -filter:a "volume=2" AUDIO.mp3`. Replace the uppercase words with the respective values. A value of `0.5` for GAIN works usually.

9.2.2 Video

Make sure the video is right side up. That is, your hands come from the bottom of the screen and play the keyboard.

If you need to rotate it, see this page for rotating with FFmpeg.

9.3 Offsets

Find the offsets for respective media. PianoRay uses these offsets. It may be beneficial to write down these offsets somewhere so you don't forget them later.

9.3.1 Audio

Open the audio in an audio player and find the timestamp, in seconds, when the audio starts. I use Audacity.

9.3.2 Video

Find the timestamp in seconds you play the first note and when you play the last note in the video. I use Blender's video editor.

9.3.3 Video Crop

Find the pixel coordinates of the four corners of the keyboard in the video, starting from the top left and going clockwise. If you use Blender's video editor, keep in mind that Blender's image viewer has the Y coordinates reversed.

9.4 Rendering

Follow instructions in this page for rendering instructions.

CHAPTER

TEN

CLI

Command line interface arguments.

Type `pianoray -h` for info.

10.1 Example Commands

- Render: `pianoray render file.py ClassName`

10.2 Resume Previous Render

While rendering, PianoRay saves which frame is currently being rendered to the cache. This allows resuming a render if it is interrupted.

Configure render resuming with the `--resume=...` flag.

- If omitted, PianoRay will ask via stdin if you wish to resume.
- If `True`, PianoRay will always resume if a previous render exists.
- If `False`, PianoRay will never resume.

If the previous render finished completely, you can pass `--resume=True` to only recompile the frames into a video.

CHAPTER

ELEVEN

API

The API is exposed as a Python module, `pianoray`.

This page contains documentation for each object. See *Animation* for information on how to use the API to animate.

11.1 Property Group

class pianoray.**PropertyGroup**

Group of properties. Define a subclass to create your PropertyGroup. Define properties by creating annotations with `:`. Don't override any methods, as instancing a PropertyGroup subclass requires the methods.

```
class MyProps(PropertyGroup):
    temperature: FloatProp(
        name="Temperature",
        desc="Temperature to cook the food at.",
        default=-10,
    )

    food: StringProp(
        name="Food",
        desc="The food to cook.",
        default="Java",
    )
```

You can set and get properties.

```
pgroup.temperature                # Returns the property object.
pgroup.temperature.animate(...)   # Animate. See Property docs.
pgroup.temperature = -273         # Calls pgroup.temperature.set_value()
```

11.2 Properties

class pianoray.**Property**(*name: str = ''*, *desc: str = ''*, *animatable: bool = True*, *required: bool = True*, *mods: Sequence[Modifier] = ()*, *default: Optional[Any] = None*)

Property base class.

animate(**args*) → None

Insert a keyframe.

A few syntaxes are available:

```
prop.animate(Keyframe(frame, value, interp))
prop.animate(Keyframe(frame, value, interp), Keyframe(frame, value, interp))
prop.animate(frame, value, interp)
prop.animate((frame, value, interp))
prop.animate((frame, value, interp), (frame2, value2, interp2), ...)
```

They all do the same thing. However, please do not mix syntaxes in one call (don't pass a keyframe object and then an unpacked tuple).

set_value(*value: Any*)

Checks validity and sets self._value

value(*frame: int*, *use_mods: bool = True*, *default: Optional[Accessor] = None*) → Any

Returns value at frame. Uses keyframe interpolations. Converts to type. Applies modifiers.

verify(*value: Any*) → bool

Check whether the value can be assigned to this prop, e.g. min and max.

Default implementation returns True. Override in subclass, if applicable.

class pianoray.**BoolProp**(*name: str = ''*, *desc: str = ''*, *animatable: bool = True*, *required: bool = True*, *mods: Sequence[Modifier] = ()*, *default: Optional[Any] = None*)

Boolean.

type

alias of `bool`

class pianoray.**IntProp**(*min: Optional[int] = None*, *max: Optional[int] = None*, *coords: bool = False*, ***kwargs*)

Integer. Min and max inclusive. Coords: Whether this quantity is in coords.

type

alias of `int`

verify(*value: int*) → bool

Checks min and max.

class pianoray.**FloatProp**(*min: Optional[float] = None*, *max: Optional[float] = None*, *coords: bool = False*, ***kwargs*)

Float. Min and max inclusive. Coords: Whether this quantity is in coords.

type

alias of `float`

verify(*value: float*) → bool

Checks min and max.

class pianoray.**StrProp**(*min_len: Optional[int] = None*, *max_len: Optional[int] = None*, ***kwargs*)

String. Min and max inclusive.

type

alias of `str`

verify(*value: str*) → bool

Checks length min and max.

class pianoray.**PathProp**(*isfile: bool = False*, *isdir: bool = False*, ***kwargs*)

Path property. Can verify if a path exists.

verify(*value: str*) → bool

Checks path isfile and isdir, if respective attributes are True.

class pianoray.**ArrayProp**(*shape: Optional[Tuple[int]] = None*, ***kwargs*)

Numpy array property.

verify(*value: ndarray*) → bool

Checks shape.

class pianoray.**RGBProp**(***kwargs*)

RGB color property.

11.3 Scene

class pianoray.**Scene**

Group of PropertyGroups.

Create a subclass with your pgroups. Set the dictionary `_pgroups` to mapping of id to property group instance.

```
class MyScene(Scene):
    _pgroups = {
        "food": FoodProps(),
    }
```

Create a subclass of a scene, and override the setup method to do animation. Scene.setup is called at initialize time.

```
# We are extending "MyScene", described above.
class MyOtherScene(MyScene):
    def setup(self):
        self.food.temperature = 100
```

property default: Accessor

Equivalent to `self.values(0)`. Usually used to get non animatable props.

setup() → None

Do any animation or property value setting here.

values(*frame: int*, *use_mods: bool = True*) → Accessor

Returns Accesor object of all pgroup values at frame.

CHAPTER

TWELVE

PROPERTIES

Automatically generated property docs.

12.1 AudioProps

- **scene.audio.file:** *PathProp*

  Name
  Audio File

  Description
  Path to audio file.

  Animatable
  `False`

  Required
  `True`

  Is File
  `True`

- **scene.audio.start:** *FloatProp*

  Name
  Start Time

  Description
  Timestamp, in seconds, you press the first note.

  Animatable
  `False`

  Required
  `True`

  Default
  `0.0`

12.2 BlocksProps

- **scene.blocks.speed:** *FloatProp*

  Name
  Speed

  Description
  If `X` is the distance between the top of the screen and thetop of the keyboard, the blocks travel `speed * X` per second.

  Animatable
  `True`

  Required
  `True`

  Default
  `0.42`

- **scene.blocks.color:** *RGBProp*

  Name
  Color

  Description
  Color of the blocks.

  Animatable
  `True`

  Required
  `True`

  Default
  `[0.6 0.65 0.9 ]`

- **scene.blocks.radius:** *FloatProp*

  Name
  Corner Radius

  Description
  Corner rounding radius in coords.

  Animatable
  `True`

  Modifiers
  `Coords`

  Required
  `True`

  Default
  `0.25`

  Minimum
  `0`

- **scene.blocks.bottom_glow:** *FloatProp*

  Name
  Bottom Glow

Description
Intensity multiplier of block glow when it hits the keyboard.

Animatable
`True`

Required
`True`

Default
`4.0`

Minimum
`0`

- **`scene.blocks.bottom_glow_len`:** *FloatProp*

  Name
  Bottom Glow Length

  Description
  Amount of bottom glow in coords.

  Animatable
  `True`

  Modifiers
  `Coords`

  Required
  `True`

  Default
  `2.0`

12.3 CompositingProps

- **`scene.comp.margin_start`:** *FloatProp*

  Name
  Start Margin

  Description
  Pause, in seconds, before first note starts.

  Animatable
  `False`

  Required
  `True`

  Default
  `3.0`

  Minimum
  `0`

- **`scene.comp.margin_end`:** *FloatProp*

  Name
  End Margin

Description
Pause, in seconds, after the last note ends.

Animatable
`False`

Required
`True`

Default
`3.0`

Minimum
`0`

- **scene.comp.fade_in:** *FloatProp*

  Name
  Fade In

  Description
  Seconds of fade in.

  Animatable
  `False`

  Required
  `True`

  Default
  `1.0`

  Minimum
  `0`

- **scene.comp.fade_out:** *FloatProp*

  Name
  Fade Out

  Description
  Seconds of fade out.

  Animatable
  `False`

  Required
  `True`

  Default
  `1.0`

  Minimum
  `0`

- **scene.comp.fade_blur:** *FloatProp*

  Name
  Fade Blur

  Description
  Blur radius of fade in coords.

  Animatable
  `False`

Modifiers
`Coords`

Required
`True`

Default
`1.0`

- **`scene.comp.shutter`:** *`FloatProp`*

  Name
  Shutter

  Description
  Shutter (brightness), lower = dimmer.

  Animatable
  `True`

  Required
  `True`

  Default
  `1.2`

  Minimum
  `0`

12.4 GlareProps

- **`scene.glare.radius`:** *`FloatProp`*

  Name
  Radius

  Description
  Radius of glare in coords.

  Animatable
  `True`

  Modifiers
  `Coords`

  Required
  `True`

  Default
  `3.0`

  Minimum
  `0`

- **`scene.glare.intensity`:** *`FloatProp`*

  Name
  Intensity

  Description
  Intensity of glare.

Animatable
True

Required
True

Default
0.9

Minimum
0

- **scene.glare.jitter:** *FloatProp*

  Name
  Jitter

  Description
  Range of random multiplier.

  Animatable
  True

  Required
  True

  Default
  0.08

  Minimum
  0

- **scene.glare.streaks:** *IntProp*

  Name
  Streaks

  Description
  Number of streaks.

  Animatable
  True

  Required
  True

  Default
  6

  Minimum
  0

  Maximum
  20

12.5 KeyboardProps

- **scene.keyboard.file:** *PathProp*

  Name
  Video File

  Description
  Path to video recording of keyboard.

  Animatable
  `False`

  Required
  `True`

  Is File
  `True`

- **scene.keyboard.start:** *FloatProp*

  Name
  Start

  Description
  Timestamp, in seconds, when the first note starts in the video.

  Animatable
  `False`

  Required
  `True`

- **scene.keyboard.end:** *FloatProp*

  Name
  End

  Description
  Timestamp, in seconds, when the last note starts in the video.

  Animatable
  `False`

  Required
  `True`

- **scene.keyboard.crop:** *ArrayProp*

  Name
  Crop

  Description
  Crop points of the keyboard. See docs for more info.

  Animatable
  `False`

  Required
  `True`

- **scene.keyboard.dim_mult:** *FloatProp*

Name
Multiplicative Dimming

Description
Multiplier to pixel brightness.

Animatable
`True`

Required
`True`

Default
`1.0`

Minimum
`0`

- **scene.keyboard.dim_add:** *FloatProp*

  Name
  Additive Dimming

  Description
  Value added to pixel brightness (0 to 255).

  Animatable
  `True`

  Required
  `True`

  Default
  `0.0`

- **scene.keyboard.below_length:** *FloatProp*

  Name
  Length of Below Section

  Description
  Length in coords of section below keyboard.

  Animatable
  `True`

  Modifiers
  `Coords`

  Required
  `True`

  Default
  `7.0`

  Minimum
  `0`

- **scene.keyboard.octave_lines:** *BoolProp*

  Name
  Octave Lines

  Description
  Whether to render octave lines.

Animatable
`True`

Required
`True`

Default
`True`

12.6 MidiProps

- **`scene.midi.file`:** *PathProp*

  Name
  MIDI File

  Description
  Path to MIDI file.

  Animatable
  `False`

  Required
  `True`

  Is File
  `True`

- **`scene.midi.speed`:** *FloatProp*

  Name
  Speed Multiplier

  Description
  MIDI notes speed multiplier.

  Animatable
  `False`

  Required
  `True`

  Default
  `1.0`

- **`scene.midi.min_length`:** *FloatProp*

  Name
  Minimum Duration

  Description
  Min duration of a note in seconds.

  Animatable
  `False`

  Required
  `True`

  Default
  `0.1`

Minimum
0

12.7 PianoProps

- **scene.piano.black_width_fac:** *FloatProp*

  Name
  Black Key Width Factor

  Description
  Black key width as factor of white key width.

  Animatable
  False

  Required
  True

  Default
  0.6

  Minimum
  0

12.8 ParticleProps

- **scene.ptcls.pps:** *FloatProp*

  Name
  Particles per Second

  Description
  Number of particles to emit per note per second.

  Animatable
  True

  Required
  True

  Default
  40.0

  Minimum
  0

- **scene.ptcls.air_resist:** *FloatProp*

  Name
  Air Resistance

  Description
  Velocity multiplies by this every second.

  Animatable
  True

Required
`True`

Default
`0.6`

Minimum
`0`

- **scene.ptcls.lifetime:** *FloatProp*

  Name
  Lifetime

  Description
  Particle lifetime in seconds.

  Animatable
  `True`

  Required
  `True`

  Default
  `3.0`

  Minimum
  `0`

- **scene.ptcls.x_vel:** *FloatProp*

  Name
  X Velocity

  Description
  Initial X velocity range in coords/sec.

  Animatable
  `True`

  Modifiers
  `Coords, SecToFrame`

  Required
  `True`

  Default
  `1.0`

  Minimum
  `0`

- **scene.ptcls.y_vel:** *FloatProp*

  Name
  Y Velocity

  Description
  Initial Y velocity range in coords/sec.

  Animatable
  `True`

  Modifiers
  `Coords, SecToFrame`

Required
True

Default
4.0

Minimum
0

- **scene.ptcls.wind_strength:** *FloatProp*

  Name
  Wind Strength

  Description
  Strength multiplier of wind affecting particles.

  Animatable
  True

  Required
  True

  Default
  1.0

- **scene.ptcls.heat_strength:** *FloatProp*

  Name
  Heat Strength

  Description
  Strength multiplier of heat affecting particles.

  Animatable
  True

  Required
  True

  Default
  1.0

- **scene.ptcls.gravity:** *FloatProp*

  Name
  Gravity

  Description
  Strength multiplier of gravity affecting particles.

  Animatable
  True

  Required
  True

  Default
  1.0

12.9 VideoProps

- **scene.video.resolution:** *ArrayProp*

  Name
  Resolution

  Description
  Output video resolution.

  Animatable
  `False`

  Required
  `True`

  Default
  `[1920 1080]`

- **scene.video.fps:** *IntProp*

  Name
  FPS

  Description
  Frames per second of output video.

  Animatable
  `False`

  Required
  `True`

  Default
  `30`

  Minimum
  `1`

- **scene.video.vcodec:** *StrProp*

  Name
  Video Codec

  Description
  Codec for video, passed to FFmpeg.

  Animatable
  `False`

  Required
  `True`

  Default
  `libx265`

CHAPTER

THIRTEEN

SETUP

How to setup your development environment.

13.1 Dependencies

See dependencies in *Installation*.

Additional dependencies for development:

- Git
- GNU Make

13.2 Fork and Clone

First, fork the GitHub repository and clone your fork.

13.3 Test Video

```
cd /path/to/pianoray
make wheel
make install
pianoray render tests/furelise.py FurElise -o out.mp4 -p
```

This should render the video and open it in your video player. Rendering may take a few minutes.

CHAPTER

FOURTEEN

CONVENTIONS

Conventions used internally.

- Frame zero is when the first note begins.
- Note zero is lowest note on piano.
- One `coord` (unit of distance) is the width of one white key in the video. This is equal to the horizontal resolution divided by 52. For example, for a 1920x1080 video, one coord is `36.924` pixels.
- C++ functions called by Python may take many arguments in order to obtain all required prop values. The naming convention is `p_video_fps` or `d_img` or `dp_blocks_color`. `p` means a property. `d` means raw data (numpy array pointer) which will be wrapped with an internal class.

CHAPTER

FIFTEEN

RENDERING

Description of how rendering is done internally.

15.1 Pipeline

First, an image of 64-bit floats is created. This is like an unbounded brightness image of the rendered scene, and will be converted into a standard 8-bit int later.

Each effects is applied to the image.

Last, the compositing library processes the double image, such as adding glare. After everything is finished, the float image is converted into an int image using `tanh` as the transformation function.

CHAPTER

SIXTEEN

C INTEGRATION

Description of how C libraries are integrated with Python.

All C code is in `pianoray/cutils`.

Code that calls the compiler is in `pianoray/cpp.py`.

16.1 Compilation

At every run, the libraries are compiled and stored in the cache directory (default `.prcache`).

16.2 Loading

Libraries are compiled to shared libraries (`.so`) and loaded with the Python `ctypes` module.

16.3 Conventions

Images are of shape `(height, width, 3)` and type `uint8` and `double`. See *Rendering* for more info on how rendering is done.

Parsed MIDI notes (start, end, note, velocity) are serialized as a string Python side and parsed C side in order to reduce the amount of function arguments (one `char*` vs four `double*`). Serialization specification can be found in *Specifications*.

CHAPTER

SEVENTEEN

SPECIFICATIONS

Specs of internally used protocols.

17.1 MIDI Notes

Notes are parsed from a MIDI file using the Python module `mido`. In order to simplify passing these notes to C functions, we serialize them into a string Python side and parse them C side. This reduces the amount of arguments required for a C function and removes boilerplate code.

Each note is stored internally as four values, `(start_frame, end_frame, note, velocity)`. The serialized string representing a sequence of notes is as follows:

```
uint32 (4bytes): How many notes there are.
For each note:
    double (8bytes): Start frame.
    double (8bytes): End frame.
    uint8 (1byte): Note index.
    uint8 (1byte): Velocity.
```

CHAPTER

EIGHTEEN

FILE STRUCTURE

Information about the project files.

18.1 /

Root directory. Contains cool files.

Python module `setup.py` and `MANIFEST.in` are here.

There is a Makefile with convenient targets.

18.2 .github

GitHub files, like workflows.

18.3 docs

Documentation. Docs are generated with Python sphinx and hosted on ReadTheDocs.

18.4 examples

Example recordings and renders.

18.5 pianoray

Source code for everything.

18.5.1 pianoray/

Main module and global utilities. Entry point is here (`__main__`).

18.5.2 pianoray/cutils

C++ libraries for rendering.

18.5.3 pianoray/effects

OOP effects for organization. Most call libraries from `cutils`.

18.5.4 pianoray/render

Rendering pipeline.

18.5.5 pianoray/view

PianoRay viewer. Currently in development.

18.6 scripts

Small scripts, like style checks.

18.7 tests

Testing files, like test video settings.

CHAPTER

NINETEEN

CACHE

PianoRay stores temporary files in a cache directory (default `.prcache`). The cache can be safely deleted at any time.

19.1 File Structure

- `./c_libs`: Compiled C library object and library files.
- `./output`: Output render is stored here.
- `./settings.json`, `./currently_rendering.txt`: Files that store the state of the rendering. This is used to resume rendering if desired. See CLI for more info.

INDEX