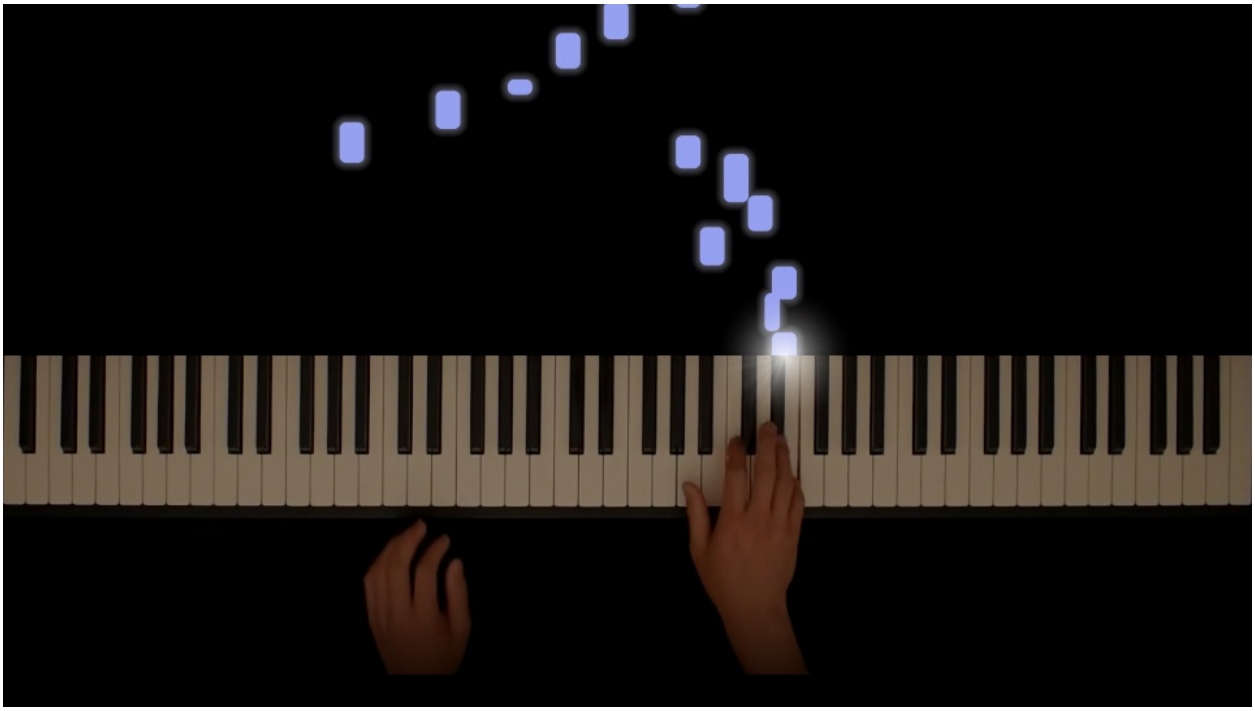
PianoRay

Release 0.2.0

PianoRay Authors

May 07, 2022

GENERAL

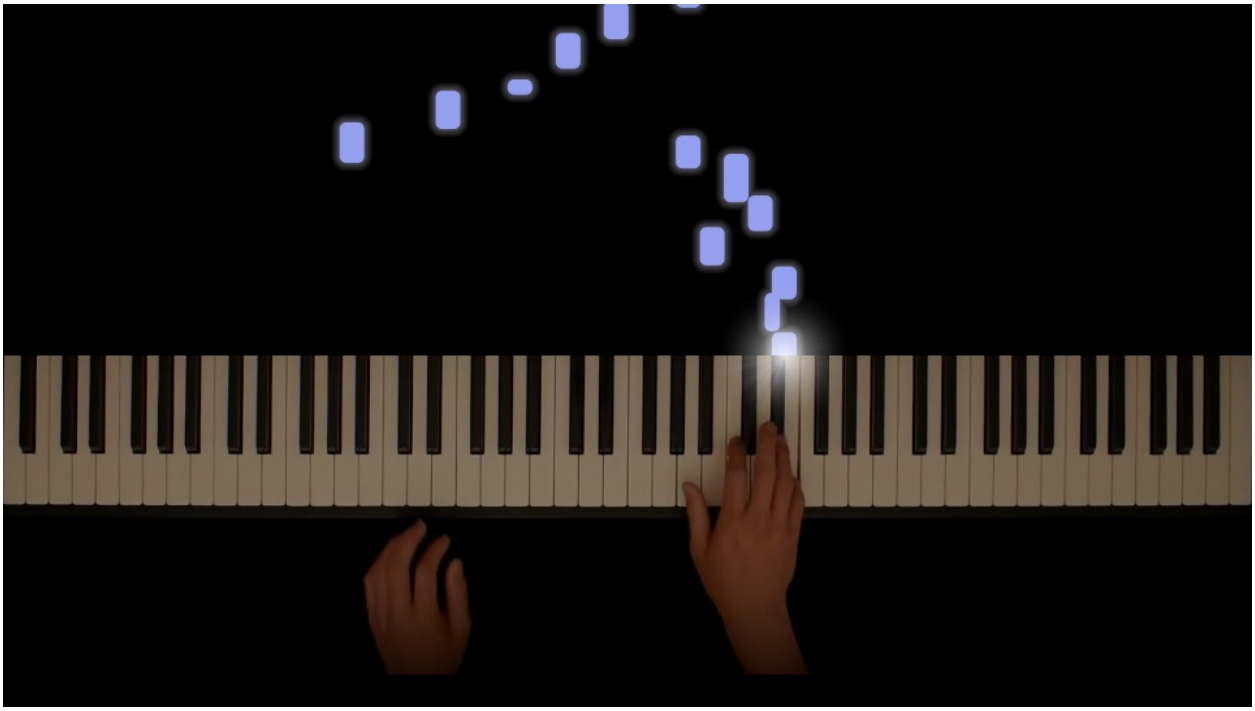


CHAPTER

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

INSTALLATION

2.1 Dependencies

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

2.2 Latest

PianoRay is distributed on PyPI. Install with

```
pip install pianoray
```

2.3 Master Branch

May be unstable or have bugs.

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

CHAPTER

THREE

SUPPORT

Please open an issue or discussion on GitHub.

CHAPTER

FOUR

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

FIVE

FIRST VIDEO

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

5.1 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.

5.2 Create Settings

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

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 = "examples/furelise/midi.mid"

        self.audio.file = "examples/furelise/audio.mp3"
        self.audio.start = 20.74

        self.keyboard.file = "examples/furelise/video.mp4"
        self.keyboard.start = 4.75
        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.

5.3 Render

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

```
cd ~/pianoray_tutorial
pianoray render furelise.py FurElise -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

SIX

RECORDING

Instructions for recording and making your own video.

6.1 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.

6.1.1 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.

6.1.2 MIDI

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

6.2 Processing

6.2.1 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.

6.2.2 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.

6.3 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.

6.3.1 Audio

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

6.3.2 Video

Find the moment you play the first note in the video. I use Blender's video editor.

6.3.3 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.

6.4 Rendering

Follow instructions in this page for rendering instructions.

CHAPTER

SEVEN

CLI

Command line interface arguments.

Type `pianoray -h` for info.

7.1 Example Commands

- Render: `pianoray render file.py ClassName`

7.2 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

EIGHT

API

The API is exposed as a Python module, `pianoray`. This page contains documentation for each object. See TODO for information on how to use the API.

8.1 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()
```

8.2 Properties

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

Property base class.

set_value(*value: Any*)

Checks validity and sets self._value

value(*frame: int*, *use_mods: bool = True*, *default: Optional[pianoray.api.accessor.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*, *mods: Sequence[pianoray.api.modifiers.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: numpy.ndarray*) → bool

Checks shape.

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

RGB color property, 0 to 255.

CHAPTER

NINE

PROPERTIES

Automatically generated property docs.

9.1 AudioProps

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

  Name Audio File

  Description Path to audio file.

  Animatable `False`

  Modifiers `[]`

  Is File `True`

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

  Name Start Time

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

  Animatable `False`

  Modifiers `[]`

  Default `0.0`

9.2 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`

  Modifiers `[]`

  Default `0.5`

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

  Name Color

Description Color of the blocks.

Animatable `True`

Modifiers `[]`

Default `[150 160 240]`

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

  Name Corner Radius

  Description Corner rounding radius in coords.

  Animatable `True`

  Modifiers `['Coords']`

  Default `0.25`

  Minimum `0`

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

  Name Glow Intensity

  Description Intensity of glow around blocks.

  Animatable `True`

  Modifiers `[]`

  Default `0.3`

  Minimum `0`

- **`scene.blocks.glow_color`:** *RGBProp*

  Name Glow Color

  Description Color of the glow.

  Animatable `True`

  Modifiers `[]`

  Default `[230 230 255]`

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

  Name Glow Radius

  Description Radius of glow around blocks in coords.

  Animatable `True`

  Modifiers `['Coords']`

  Default `0.4`

9.3 CompositionProps

- **scene.composition.margin_start:** *FloatProp*

  Name Start Margin

  Description Pause, in seconds, before first note starts.

  Animatable `False`

  Modifiers `[]`

  Default `3.0`

  Minimum `0`

- **scene.composition.margin_end:** *FloatProp*

  Name End Margin

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

  Animatable `False`

  Modifiers `[]`

  Default `3.0`

  Minimum `0`

- **scene.composition.fade_in:** *FloatProp*

  Name Fade In

  Description Seconds of fade in.

  Animatable `False`

  Modifiers `[]`

  Default `1.0`

  Minimum `0`

- **scene.composition.fade_out:** *FloatProp*

  Name Fade Out

  Description Seconds of fade out.

  Animatable `False`

  Modifiers `[]`

  Default `1.0`

  Minimum `0`

- **scene.composition.fade_blur:** *FloatProp*

  Name Fade Blur

  Description Blur radius of fade in coords.

  Animatable `False`

  Modifiers `['Coords']`

  Default `1.0`

9.4 GlareProps

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

  Name Radius

  Description Radius of glare in coords.

  Animatable `True`

  Modifiers `['Coords']`

  Default `3.0`

  Minimum `0`

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

  Name Intensity

  Description Intensity of glare.

  Animatable `True`

  Modifiers `[]`

  Default `0.9`

  Minimum `0`

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

  Name Jitter

  Description Range of random multiplier.

  Animatable `True`

  Modifiers `[]`

  Default `0.08`

  Minimum `0`

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

  Name Streaks

  Description Number of streaks.

  Animatable `True`

  Modifiers `[]`

  Default `6`

  Minimum `0`

  Maximum `20`

9.5 KeyboardProps

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

  Name Video File

  Description Path to video recording of keyboard.

  Animatable `False`

  Modifiers `[]`

  Is File `True`

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

  Name Start

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

  Animatable `False`

  Modifiers `[]`

  Default `0.0`

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

  Name Crop

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

  Animatable `False`

  Modifiers `[]`

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

  Name Multiplicative Dimming

  Description Multiplier to pixel brightness.

  Animatable `True`

  Modifiers `[]`

  Default `1.0`

  Minimum `0`

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

  Name Additive Dimming

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

  Animatable `True`

  Modifiers `[]`

  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']`

Default `7.0`

Minimum `0`

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

  Name Octave Lines

  Description Whether to render octave lines.

  Animatable `True`

  Modifiers `[]`

  Default `True`

9.6 MidiProps

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

  Name MIDI File

  Description Path to MIDI file.

  Animatable `False`

  Modifiers `[]`

  Is File `True`

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

  Name Speed Multiplier

  Description MIDI notes speed multiplier.

  Animatable `False`

  Modifiers `[]`

  Default `1.0`

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

  Name Minimum Duration

  Description Min duration of a note in seconds.

  Animatable `False`

  Modifiers `[]`

  Default `0.08`

  Minimum `0`

9.7 PianoProps

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

  Name Black Key Width Factor

  Description Black key width as factor of white key width.

  Animatable `False`

  Modifiers `[]`

  Default `0.6`

  Minimum `0`

9.8 VideoProps

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

  Name Resolution

  Description Output video resolution.

  Animatable `False`

  Modifiers `[]`

  Default `[1920 1080]`

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

  Name FPS

  Description Frames per second of output video.

  Animatable `False`

  Modifiers `[]`

  Default `30`

  Minimum `1`

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

  Name Video Codec

  Description Codec for video, passed to FFmpeg.

  Animatable `False`

  Modifiers `[]`

  Default `libx265`

CHAPTER

TEN

SETUP

How to setup your development environment.

10.1 Dependencies

See dependencies in the `General/Installation` page.

Additional dependencies for development:

- Git
- GNU Make

10.2 Fork and Clone

First, fork the GitHub repository and clone your fork.

10.3 Test Video

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

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

CHAPTER

ELEVEN

FILE STRUCTURE

Information about the project files.

11.1 /

Root directory. Contains cool files.

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

There is a Makefile with convenient targets.

11.2 .github

GitHub files, like workflows.

11.3 docs

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

11.4 examples

Example recordings for testing.

11.5 pianoray

Source code for everything.

11.5.1 pianoray/

Main module and global utilities.

- `__init__.py`: Module file.
- `cpp.py`: Handles C++ library compiling and loading.
- `logger.py`: Logging utilities.
- `main.py`: Main entry point.
- `settings.py`: Class for convenient settings access.
- `utils.py`: Global utilities.

11.5.2 pianoray/cutils

C++ header files for C++ libraries.

- `pr_image.hpp`: Image class for interacting with raw unsigned char data.
- `pr_math.hpp`: Math utilities.
- `pr_piano.hpp`: Utilities relating to rendering, e.g. piano dimensions.
- `pr_random.hpp`: Random number generator utilities.

11.5.3 pianoray/effects

Files that render the video.

- `effect.py`: Effect base class for OOP internally.
- `midi.py`: Parse MIDI.
- `blocks.py`, `blocks.cpp`: Rendering blocks.
- `glare.py`, `glare.cpp`: Rendering glare.
- `keyboard.py`: Rendering keyboard.

11.5.4 pianoray/render

Rendering pipeline.

- `render.py`: Calling effects to render the video.
- `video.py`: Class for managing video frames and calling FFmpeg to compile the video.
- `lib.py`: Load and initialize C++ libraries.

11.5.5 pianoray/view

PianoRay viewer files. Currently in development.

11.6 scripts

Small scripts, like style checks.

11.7 tests

Testing files, like test JSON settings.

CHAPTER

TWELVE

CACHE

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

12.1 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.

CHAPTER

THIRTEEN

INFORMATION

Just information for now.

- 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.

INDEX