Vision-based beatmap extraction in rhythm game toward platform-aware note generation

Yeonghun Kim
School of Computing
Korea Advanced Institute of Science and Technology
Republic of Korea
neutrinoant@kaist.ac.kr

Sunghee Choi
School of Computing
Korea Advanced Institute of Science and Technology
Republic of Korea
sunghee@kaist.edu

***Abstract*—Recent approaches to deep learning-based music analysis have had significant impact on procedural content generation in music-based games. However, the lack of understanding of the unique features of various platforms and interfaces makes auto-generated content less valuable than manually designed content. Hand-crafted datasets are required, to enhance the quality of content in various platforms, but most rhythm games permit only indirect access to the dataset, as a form of player's experience and its replay video. We develop a vision-based approach to content extraction through video analysis, using a format named *beatmap*. We cover some common visualized features in well-known rhythm games, and construct a mapping from their content to our beatmap model, using multiple object detection. Our method correctly detects each action button, type, and time, and extracts beatmap representations for our target game.**

***Index Terms*—procedural content generation, music-based game, video analysis, object detection, beatmap extraction**

I. Introduction

Procedural content generation (PCG) has become a popular research topic in a range of games, including the music-based game called the rhythm game [1] [2]. The rhythm game has common concepts, played by listening to music and taking appropriate action at a specific times. One of the most popular designs, which we call *keyboard-type*, involves an object *note*, moving along a fixed route. When the note arrives at specific location, the user takes an action corresponding to the route. Many well-known games adopt this type of design, and have been widely researched on various platforms, including osu! [2] on the PC, BeatSaber [3] on VR devices, and Dance Dance Revolution [4] on arcade machines.

In rhythm game design, the manual design of notes is quite demanding. Naively, the content designer needs to select proper beats synced with the music, and more importantly needs to fully understand the interface used, such as the number of buttons, their shapes, sizes, and even locations per route. Various platforms have unique interfaces involving more than just keyboard buttons, and each interface includes underlying human-centric rules such that 10 fingers cannot press 11 buttons together, or two feet cannot reach two buttons together more than a leg length apart. In this context, the players' experience of the interface is crucial for naturalness and for the generation of a high level of entertainment.

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

Recent work into deep learning has raised the possibility of producing high performance and scalability of PCG of rhythm games, using automatic note generation. Yubin's team [1] trained their learning model using the osu! beatmap dataset to generate note data, such as the action time and button to be pressed, and the model allows any kind of audio as input. However, the dataset depends strongly on the keyboard interface, so the generated content does not fit other devices with different button structures. Hand-crafted datasets can help to overcome this limitation and facilitate further research, but most rhythm games permit only indirect access to the dataset as a form of the player's experience and its recorded video.

With this motivation, we develop a system for vision-based automatic note extraction from replay video to build rhythm game datasets on various platforms. Our method utilizes multiple object detection (MOD) to detect notes on each frame, using two-pass video scanning. In the first pass, we analyze the representation of moving notes, such as notes coming toward the screen, falling like a waterfall, or other variations. In the second pass, we use MOD to find notes in each frame with respect to the note representation, and merge them into a single rectangular model, which we define as a *beatmap*. Since the beatmap comes from hand-crafted data from the game, it reflects both the correct rhythm of the music, and an understanding of the underlying interface structure.

There is a similar but different attempt that considers the game interface in PCG. Azizah's team [5] points out that content obtained only from music analysis is not applicable to VR, since it does not reflect the spatial properties of the player's actions. Their solution utilizes human dance to reflect the unique features of the VR interface and enhance the quality of note generation. While they target only VR devices, we target mostly arcade machines equipped with their own unique interfaces, and even mobile devices and PC.

II. Methodology

A. The Beatmap Model

The scope of our work is bounded on keyboard-type 2D rhythm games, as shown in Fig. 1, as follows. There is an infinitely long road partitioned into multiple *routes* and a

Fig. 1. Examples of keyboard-type rhythm games: O2Jam (top-left), Beatmania IIDX (top-right), Deemo (bottom-left), and Nostalgia (bottom-right). Four games have common features, including multiple tracks and falling notes along the tracks from top to bottom, but they have different ways of visualization of notes.

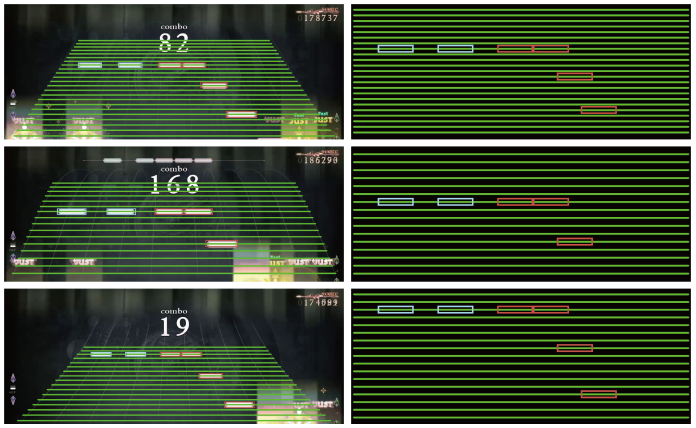

Fig. 2. Figure 2. Three examples of line-to-line mapping between non-uniform frame coordinates and uniform rectangular coordinates. Left figures are the game screenshots applying visual options in our target game, named Approach (row 1), Splash (row 2), and Slope (row 3). Right figures are the corresponding results of inverse warping of the bar lines (green) and notes detected.

judgement line at a fixed position on the road. On one side of the road, *notes* run toward the judgement line along their own routes, and generally their speeds do not need to be the same. A *frame window* records the scene near the line, but visualizes it with a warped vision. *Warping* in this situation includes any kind of perspective transform mapping each horizontal line segment on the road into another horizontal line segment in the frame. Using these descriptions, our goal is to extract each note on a video as a triple (w, t, k), where w is the route index, t is the arrival time to the judgement line, and k is the note type. We call the set of triples a *beatmap*; the name originates from the osu! rhythm game.

B. Inverse Warping

Warping by frame window makes note-tracking harder, since it causes each note to change its shape and speed according to its relative location on the frame. To correctly analyze the moving pattern of notes, we construct an inverse map of the warping function between two different coordinates, before and after warping. Based on our assumption in Section II-A, it is sufficient to find line-to-line correspondences on each y-coordinate. A line is mostly shown as a *bar* in rhythm games.

A bar, called a measure as a musical element, is visualized as a horizontal line in many rhythm games. We use bar lines to analyze the moving pattern of notes on the frame. Line segment detection [6] is used to detect all of the horizontal lines appearing on all of the frames, with their y-axis locations, lengths, and the instantaneous velocities $v(y)$ approximated as below:

$$v(y) = (y - y_{pre}) \cdot r \tag{1}$$

where r denotes the frame rate and y_{pre} denotes the y-coordinate of location of the same line in the previous frame. In general, the domain of v is only a subset of $[1, H]$, for frame height H. To expand the domain, we use Gaussian process regression (GPR) [7] and predict $v(y)$ in a domain $[1, H]$. Since GPR has an ability of fitting sample points to any continuous function without prior knowledge, it is suitable for covering many possible designs of falling notes and shapes of tracks.

Suppose that a line on a frame moves from $y = 1$ to h for t_h seconds, then the corresponding line on a frame window moves from $y = 1$ to y_h, which is approximated by Euler's method on a pixelated coordinate, as below:

$$y_h = H \cdot \frac{\sum_{i=1}^{h} \delta_i}{\sum_{i=1}^{H} \delta_i} \tag{2}$$

$$\delta_i = \frac{1}{v(i)} \tag{3}$$

where δ_i denotes the time taken for the line on frame to move from $y = i - 1$ to i. Similar analysis is done for the x-coordinates of lines. We first define $x_l(y)$ and $x_r(y)$ as the x-coordinates of left end and right end of the line of which y-coordinate is y, respectively, and predict them in the domain $[1, H]$ using GPR. Fixing $x'_l = 1$ as a left end and $x'_r = W$ (width of the frame window) as a right end for a line on a frame window, we obtain the line-to-line correspondence between the two coordinates directly from the three pairs: (h, y_h), $(x_l(h), x'_l)$, and $(x_r(h), x'_r)$. For simplicity, we represent each correspondence as 2×2 linear transform matrix, and cache all the matrices at runtime. Three examples of the inverse warping process are illustrated in Fig. 2.

C. Multi-Template Matching

Vision-based object detection enables the detection of notes in each frame. Previous work using deep learning-based schemes produce rapid improvements on multiple object detection. However, labeled note datasets are not sufficiently detailed to train models for our goal, and thus limit the scope of games. For simplicity and scalability, we use multi-template matching [8] with manually designed templates from each different shaped note. Each target frame and templates

are gray-scaled and matched using the normalized correlation coefficient R as below:

$$R(x,y) = \frac{\sum_{x',y'} T^w_{x,y}(x',y') \cdot I(x+x', y+y')}{\sqrt{\sum_{x',y'} T^w_{x,y}(x',y')^2 \cdot \sum_{x',y'} I(x+x', y+y')^2}} \tag{4}$$

$$T^w_{x,y} = L_{x,y}(T) \tag{5}$$

where T and I denote the template and target image, respectively, both normalized to zero mean. Each coefficient for (x, y) on the given frame I is calculated by scanning all the points (x', y') on the warped template $T^w_{x,y}$, which determines similarity between the template and the patch compared in the frame. $L_{x,y}$ is a template image transformation on an (x, y) location derived from the inverse warping function described in Section II-B. For our target game, we choose $L_{x,y} = s(y)$ for scaling function s, to transform each note shape, but in general L can be any other function that entirely depends on the game setting.

After processing all templates on a frame, Non-Maximum Suppression (NMS) [9] was used to identify the best matching locations of labeled bounding boxes. This step is followed by additional color-based analysis on the pixels inside each bounding box, producing each note location (x, y) and note type label k, defined in the game, including simple note, start/end of hold note, and slide note moving left to right.

D. Beatmap Construction

Using the approach described in Section II-C, we obtained points represented as (x, y, k, f), consisting of the relative note location (x, y) on the frame with bottom left coordinate as (0,0), its label, k, and the frame number, f. Our final step is to track the movement of points to extract the action time t, for each note.

To track note by note, we first set a route number, w, for each point according to game-dependent fixed ranges of routes, and group all the points by route number. For each group of points, we scan points in increasing order of y, and decide whether to add each point to a new or existing *note cluster*. A note cluster is defined as a group of points indicating the same note. Each note cluster is in a cluster queue and can get a new point (x, y, k, f) if its y is less than or equal to the minimum y of points in the cluster. A new point is added to the earliest-generated note cluster in the queue satisfying the y-coordinate condition, or otherwise is added to a new cluster in the queue. We dequeue the cluster after its corresponding note moves out of the frame rectangle. We predict the note-escaping time moment by using the note speed calculated from two points in the cluster. Note that this dequeuing process is only applied to clusters having at least two points.

After the scanning and all of the clusters are made, we use linear regression for the (y, f) pairs of points in each note cluster to predict (y_{judge}, f_{judge}). y_{judge} is the y-coordinate of the judgement line relative to the frame, and f_{judge} is the frame number at which the note arrives at the line. Since the action time, t, is equal to $f_{judge} \cdot r$ for a fixed frame rate, r, we obtain a triple (w, t, k) for each cluster, and integrate all of them into a final beatmap.

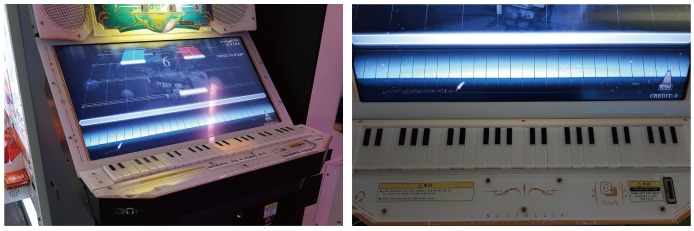

Fig. 3. The interface design of our target rhythm game, Nostalgia. The arcade machine has piano-like 28 buttons arranged in a row and has a sequential mapping between each button and route on the screen.

III. Results and Discussions

The proposed method covers most of the keyboard-type rhythm games, but still depends on a target game having unique visual effects and settings as obstacles. We targeted *Nostalgia*, a piano-based arcade game launched by KONAMI in 2017, to test our method. Fig. 3 shows the 28-key buttons interface of Nostalgia, and its features applicable to our work, summarized below.

- 28-key buttons and their corresponding routes.
- Various note falling effects: vertical, approach, splash, and slope.
- Various note types: simple, tenuto, trill, and glissando.
- Two or more notes with different colors or with inclusive relationships to each other.

The use of 28 buttons limits the maximum number of notes, and their pairwise distances needed to be handled together. For instance, a player with 10 fingers cannot handle 11 notes at the same time, or three notes located at either end and the middle. These implicit features naturally appear in the game contents, especially in the beatmaps that we aimed to extract.

We manually obtained replay video datasets by recording the output signal of the arcade machine. Each video has a fixed 60 fps frame rate and 1280×720 pixel resolution. We prepared several note template images cropped from the video with vertical note falling effects.

We first analyzed the accuracy of our method quantitatively (Table I). Ten songs with up to 1,983 notes were recorded, with three different non-vertical falling effects as options in Nostalgia. Since we have no public ground truth for songs, we first extracted beatmaps in vertical falling effects, and manually adjusted every action time per note. These machine-assisted ground truth beatmaps were compared with each effect by measuring the F1 score and mean squared error (MSE). True positives for F1 are defined in our work as all note pairs located on the same route and having at most 1/60 second of the action time difference between note pairs. For MSE, we used the action times only for true positives; the mean squared error of time differences for correctly found note pairs. The average F1 scores for 10 songs was about 0.98.

Next, we analyzed the processing time of our proposed method, step by step. Table II shows time elapsed per step for

TABLE I
AVERAGE F-SCORE AND MSE OF 10 SONGS IN DIFFERENT VISUAL EFFECTS. THREE EFFECTS ARE VISUALIZED, AS DESCRIBED IN FIG. 2.

Visual effect	F1-score	MSE $[sec^2]$
Approach	0.974	2.15×10^{-4}
Splash	0.980	2.15×10^{-4}
Slope	0.981	2.11×10^{-4}

TABLE II
ELAPSED TIME (MIN.) OF EACH STEP DESCRIBED IN SECTION II, INCLUDING CACHING. THREE EXAMPLE SONGS WERE PROCESSED: *zeeros* HAS TWO TEMPLATES AND 7,676 FRAMES; *La Campanella* HAS THREE TEMPLATES AND 7,406 FRAMES; AND *Moonlight* HAS FOUR TEMPLATES AND 8,372 FRAMES.

step	*zeeros*	*La Campanella*	*Moonlight*
Inverse Warping	0.078	0.073	0.075
Caching	5.110	4.759	5.846
Multi-template Matching	37.076	38.590	78.823
Beatmap Construction	10.598	9.596	13.711
total	52.863	53.018	98.456

three songs, using an Intel i7-8700 3.20 GHz processor and 48 GB RAM with multi-processing of 12 CPUs. To improve performance, we added another step, *Caching*, which consisted of cropping frames and caching all matrices referred to Section II-B. Multi-template matching took up the most out of the total processing time, and its per-frame time varied according to two factors. One was the large frame size, which caused a high cost for comparison, as shown in Eqn. 4, while the other was the number of templates used. Neither factor had inter-frame dependency; therefore, further speedup would be possible by adopting a GPU multi-threading scheme.

Fig. 4 shows a part of the beatmap of the song Moonlight. The vertical axis contains an integer coordinate corresponding to the frame number, closely related to the action time of the note. The input video has 8,371 frames, and our proposed method found 1,931 notes almost correctly. For this song, we used four templates with their thresholds, experimentally chosen for template matching.

A. Conclusion

We developed a vision-based automatic beatmap extraction algorithm from replay video of keyboard-type rhythm games. We tested several videos from our target game, which has a unique interface and non-vertical note falling effects. The algorithm found every note per frame, and successfully integrated them into a single beatmap. Quantitative experiments showed a high accuracy of our method, with a maximum average F-score of 0.981 and a minimum average MSE of 2.11×10^{-4} of action time differences.

Our work can be applied to many arcade, mobile, or PC rhythm games, but its performance depends on the pre- and post-processing for a target game. Visual effects and options need to be covered, as does the way in which the falling note is warped against its location. Therefore we focused on the

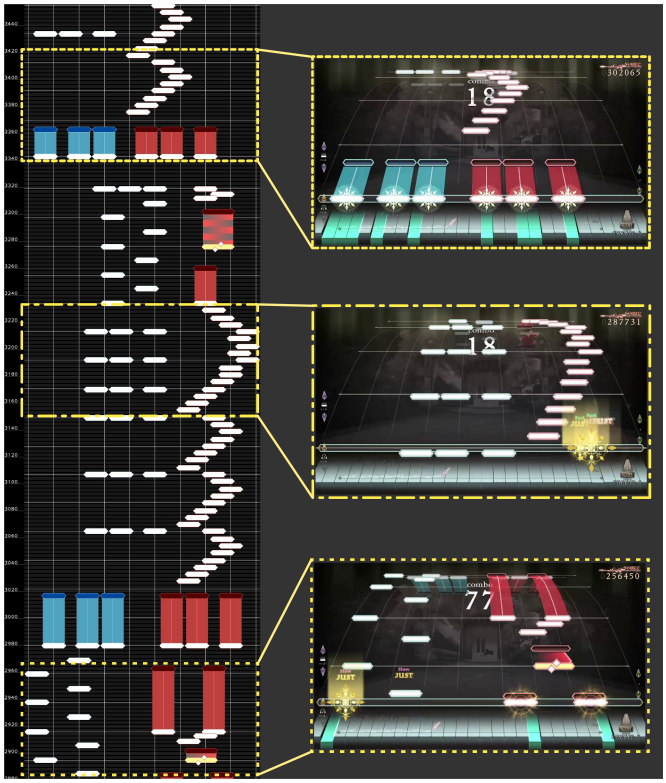

Fig. 4. An example of a beatmap extracted from the song *Moonlight* and related scenes in the video. The figure on the left visualizes various types of notes and bar lines detected in the frames on the right.

common features in well-known rhythm games, and proposed general but simple methodologies covering as many features as possible. Our work is expected to improve the quality and diversity of rhythm game datasets, and to encourage further research.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-01158, Development of a Framework for 3D Geometric Model Processing)

REFERENCES

[1] Y. Liang, W. Li, and K. Ikeda, "Procedural content generation of rhythm games using deep learning methods," in *Joint International Conference on Entertainment Computing and Serious Games*. Springer, Conference Proceedings, pp. 134–145.

[2] G. A. Salsabilla, H. Fabroyir, D. Herumurti, I. Kuswardayan, and S. C. Hidayati, "Tachyon: Multiplatform rhythm game with automatic beatmap generation," in *2020 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*. IEEE, Conference Proceedings, pp. 162–167.

[3] A. Szpak, S. C. Michalski, and T. Loetscher, "Exergaming with beat saber: An investigation of virtual reality aftereffects," *Journal of Medical Internet Research*, vol. 22, no. 10, p. e19840, 2020.

[4] C. Donahue, Z. C. Lipton, and J. McAuley, "Dance dance convolution," in *International conference on machine learning*. PMLR, Conference Proceedings, pp. 1039–1048.

[5] R. N. Azizah, Y. S. Lee, J. J. Jung, and B.-S. Sohn, "Gesture recognition for note generation in vr rhythm game," in *2021 International Conference on Information Networking (ICOIN)*. IEEE, Conference Proceedings, pp. 521–524.

[6] J. H. Lee, S. Lee, G. Zhang, J. Lim, W. K. Chung, and I. H. Suh, "Outdoor place recognition in urban environments using straight lines," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, Conference Proceedings, pp. 5550–5557.

[7] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer school on machine learning*. Springer, Conference Proceedings, pp. 63–71.

[8] G. Bradski, "The opencv library," *Dr Dobb's J. Software Tools*, vol. 25, pp. 120–125, 2000.

[9] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 3. IEEE, Conference Proceedings, pp. 850–855.