SurfChessVR: Deploying Chess Game on Parametric Surface in Virtual Reality

Wanwan Li
University of South Florida
Tampa, Florida, USA
wanwan@usf.edu

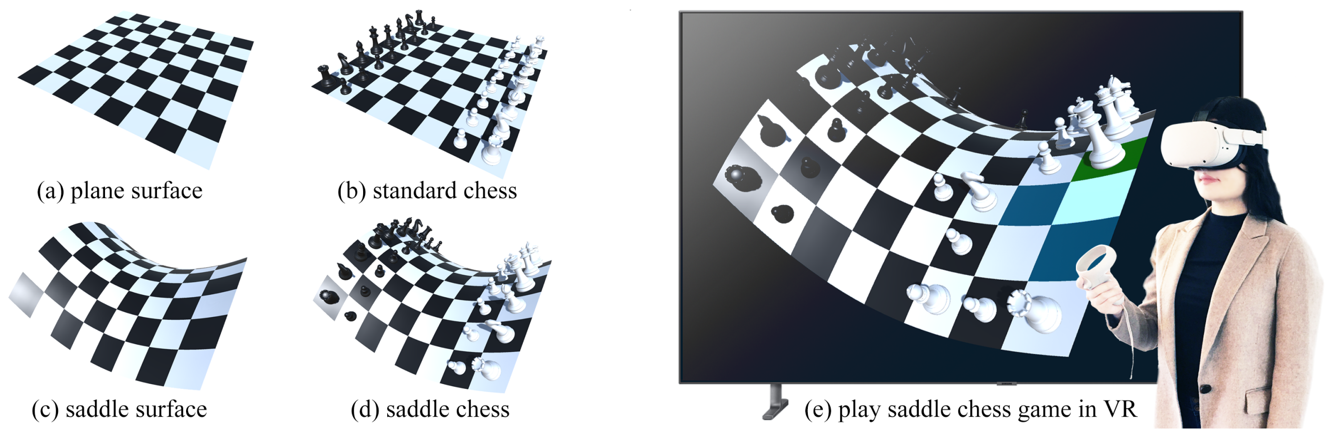

(a) plane surface (b) standard chess (c) saddle surface (d) saddle chess (e) play saddle chess game in VR

Fig. 1. This figure shows a novel gaming interface, SurfChessVR, which deploys chess games on parametric surfaces in Virtual Reality (VR). Through the parametric modeling approach proposed by us, a traditional chessboard of a plane surface (a) and (b) can be transformed into an arbitrary parametric surface chessboard automatically generated given a parametric equation (c) and (d). In this example, the parametric equation represents a saddle surface. Then, after connecting with the Unity Steam VR plugin, the player wearing Occulus Quest 2 can play this SurfChessVR game in interactive virtual environment (e).

***Abstract*—This paper proposes SurfChessVR, a novel gaming interface that deploys chess games on parametric surfaces in Virtual Reality (VR). Unlike traditional chess in VR where all chessmen are placed on a plane chessboard, our approach automatically places chessmen on arbitrary parametric surfaces such as spheres, torus, cones, etc. More specifically, given a mathematical equation that describes a parametric surface as input, our approach automatically generates the chessboard in the shape of that parametric surface and places the chessmen on that surface at different squares with corresponding orientations that align with the surface's normal directions. In order to deliver players with immersive gaming experiences of playing chess on a parametric surface, we implemented a game AI algorithm called the Min-Max algorithm that enables the computer to game against the player in VR. Experiment results and user studies validate the effectiveness and correctness of our approach.**

***Keywords*—parametric modeling, chess game, virtual reality, interactive interface, game AI, Min-Max algorithm**

I. Introduction

As one traditional abstract strategy board game between two players that involves no hidden information, chess is one of the world's most popular games and is played by millions of people worldwide today. Chess is not only a symbol of competition and intelligence, it is also a carrier that carries a boundless possibility for entertainment. New fashions in chess games attract growing interest from players with different cultural backgrounds and different ages. Especially, the traditional forms of chess will no more satisfy the young players who are from the new age. One example shows the attempts that players are trying to "renew" the concept of chess, that are, the attempts that have been made to translate a flat 8x8 board into a spherical board [1]. From time to time, people have been inspired to build spherical boards [2], round chess boards [3], cylindrical chess boards [4], etc., and figure out how to play chess on them. At an age without computers, these goals seem challenging as physically designing different styles of chess boards takes unignorable expenses, time, manual effort, and the risk caused by failures. But nowadays, with the help of computational design, this goal is realizable. Especially, given the presence of Virtual Reality (VR) technologies that deliver users immersive interactions, synthesizing chessboards in different shapes and deploying chess games on different types of chessboards becomes possible. Therefore, inspired by this trend of modern chess games, in this paper, we propose SurfChessVR, a novel gaming interface that automatically deploys chess games on parametric surfaces in VR that demands minimum manual efforts from the board game designers. Main contribution of our work include:

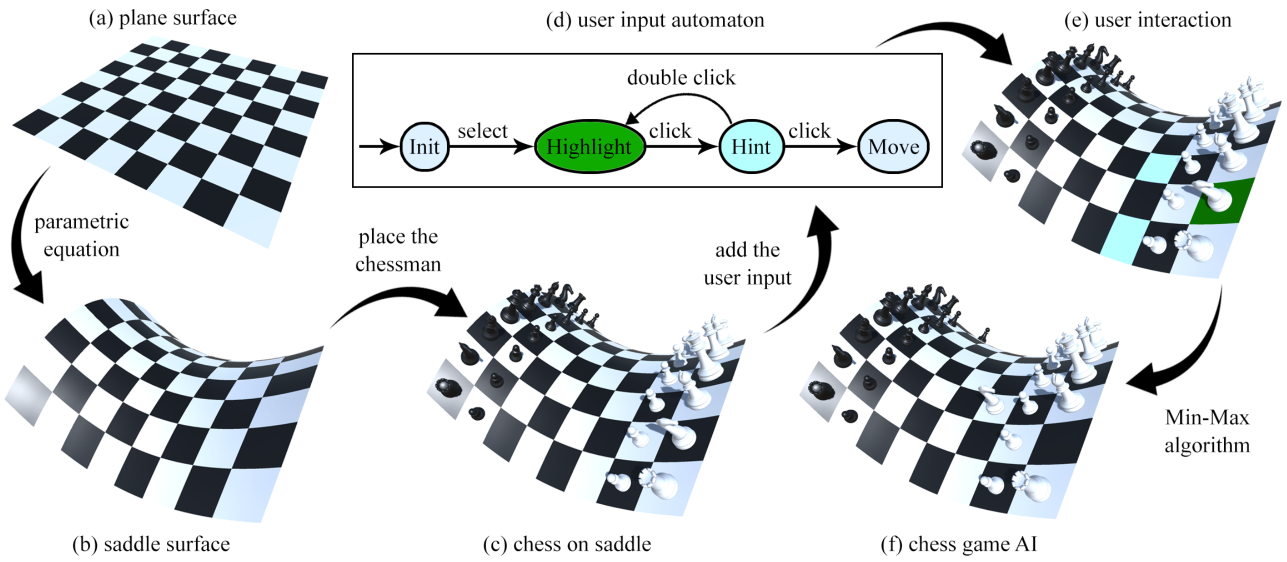


Fig. 2. Overview of our technical approach.

- we propose a novel research topic on the parametric modeling of chessboard geometry that has never been explored by other researchers before.
- we propose a novel technical approach to automatically generated the virtual chessboards with different shapes of parametric surfaces and automatically place chessmen on that parametric surface with realistic alignments.
- we conduct a series of experiments and user studies whose results validate the effectiveness and correctness of our approach. The video recording of experiments and user studies can be accessed through this URL link: https://youtu.be/wW5GcbUiQY4

II. Related Work

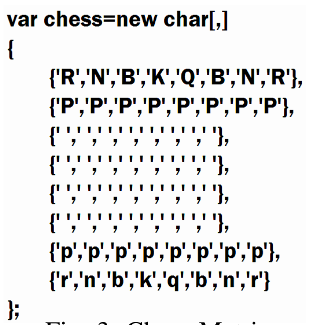
As the computational power of modern computers is growing, more and more researchers are implementing various types of chess games with robust game AI and hardware supports. In 2008, bontchev et al. [5] implemented a mobile chess game. Chen et al. [6] developed a remote Chinese chess game using mobile phone augmented reality. In 2010, Kaur et al. [7] designed and implemented an artificially intelligent microcontroller-based chess opponent. In 2012, Li et al. [8] implemented a Korean chess game by hand gesture recognition using stereo camera. In 2014, Su et al. [9] implements the chess game artificial intelligent using mobile robots. Mendes et al. [10] implemented the hardware system for automatic and interactive chess board. In 2015, Al et al. [11] designed and implemented the chess-playing robotic system. Peiravi et al. [12] designed and implemented an adaptive learning-based chess game. In 2016, Muji et al. [13] designed and implemented the electronic chess set. Wu et al. [14] designed Chinese chess algorithm and implemented the Chinese chess as a computer game. Li et al. [15] designed and implemented a personalized interface of Chinese army chess. In 2017, Kim et al. [16] deviced a real-time VR strategy chess game using motion recognition. Bhutani et al. [17] designed and implemented a wireless remote chess playing physical platform. In 2018, Larregay et al. [18] designed and implemented a computer vision system for an autonomous chess-playing robot. In 2019, Yusof et al. [19] implemented collaborative chess game in handheld devices on augmented reality platforms. Zhong et al. [20] implemented the chess game in the object-oriented programming language of C++. Shi et al. [21] designed and implemented a general chess game system client based on electron framework. Wang et al. [22] designed and implemented a Chinese chess based on robot arm manipulator. In 2020, Li et al. [23] implements the Chinese chess game algorithm based on reinforcement learning. Kolosowski et al. [24] proposed a collaborative robot system for playing chess. Zhang et al. [25] designed and implemented an intelligent Chinese chess system device. However, as far as we know, there is no existing work that has deployed chess games on parametric surface-shaped chessboards in virtual reality.

III. Overview

Figure 2 shows the overview of our technical approach. Figure 2(a) shows a traditional chess board of a plane surface. After using the parametric modeling approach proposed by us, an arbitrary surface chessboard can be automatically generated given a parametric equation as shown in Figure 2(b). In this example, input is the parametric equation of a saddle surface. Then, we propose a mathematical approach to automatically place the chessmen on the parametric surface so that the chessmen's positions align with the centers of squares and the chessmen's rotations align with the surface's normal directions as shown in Figure 2(c). After the chess game is deployed on the saddle surface through the proposed parametric modeling approach, the game starts with an initial state in the user input automaton as shown in Figure 2(d). Through this deterministic finite automaton, user interaction is added to this synthesized chess game. For example, the player can select any white chessman when the mouse (or VR controller) hangs over that chessman, and the square under that chessman will be highlighted as dark green. When the player clicks that dark green square, hints in light blue (or dark blue) will pop up to show the valid moves. If the player clicks that dark green square again, the hints will disappear. Or, if the player clicks those hints in light blue (or dark blue), that highlighted chessman will move to those hints correspondingly as shown in Figure 2(e). After the player finished each move, our proposed interface will automatically move a black chessman using a game AI algorithm as shown in Figure 2(f).

IV. Technical Approach

Representation. To simplify the chess representation, a 2D character matrix, called chess matrix, is applied to represent the current status of a game. Figure 3 shows an example of chess matrix that represents the initialization of the game. A quadruple of four numbers represents how to move the chess. For example, $(1, 7, 2, 7)$ represents a move to switch the chessman from square $[1, 7]$ to square $[2, 7]$. In this way, a linked list of quadruples can be used for storing all valid moves for the computer or player. Also, it serves as the fundamental data structure of the min-max algorithm implemented in this paper for the game AI.



```
var chess=new char[,]
{
        {'R','N','B','K','Q','B','N','R'},
        {'P','P','P','P','P','P','P','P'},
        {' ',' ',' ',' ',' ',' ',' ',' '},
        {' ',' ',' ',' ',' ',' ',' ',' '},
        {' ',' ',' ',' ',' ',' ',' ',' '},
        {' ',' ',' ',' ',' ',' ',' ',' '},
        {'p','p','p','p','p','p','p','p'},
        {'r','n','b','k','q','b','n','r'}
};
```

Fig. 3. Chess Matrix

Valid Move Check. In this work, a function is implemented to check whether it can move a chessman from one square to another square, say, they are $[i_0, j_0]$ and $[i_1, j_1]$ respectively. Firstly, check whether the chessman standing on $[i_1, j_1]$ has the same color as the one on $[i_0, j_0]$. This is simply done by checking the chess matrix that whether two characters on these two squares share the same case (upper case or lower case). The same case means the same color and turns out to be an invalid move immediately. Otherwise, according to various types of chessmen on $[i_0, j_0]$, there are various methods to continue checking whether this is a valid move correspondingly. For example, as shown in Figure 4, in this scenario, the chessman on $[i_0, j_0]$ is a Bishop 'B' which is highlighted in a green rectangle, then check whether $\Delta i = |i_1 - i_0|$ is equal to $\Delta j = |j_1 - j_0|$ which means that this move is on the diagonal. If it is diagonal, then check whether there is any chessman standing on the diagonal between $[i_0, j_0]$ and $[i_1, j_1]$. If no one is there, this is a valid move. However, a pawn 'P' is standing there which is highlighted in a red circle, so moving 'B' from $[i_0, j_0]$ to $[i_1, j_1]$ is invalid. Other checking processes act differently according to the rules of chess game.

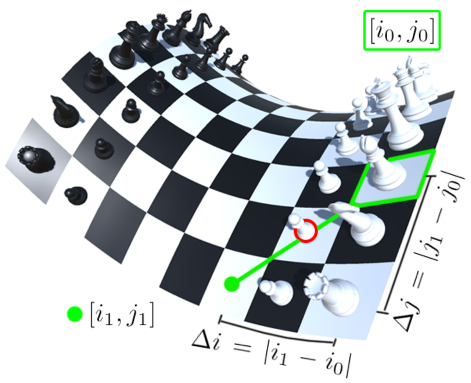


Fig. 4. Valid Move Check. This figure shows an example of checking a move is valid or not. In this case, a chessman of Bishop 'B' on $[i_0, j_0]$ (highlighted in green rectangle) can not be moved to $[i_1, j_1]$ (highlighted as a green dot) as there is a pawn (highlighted in red circle) standing in between.

Move a Chessman. Similar to the chess game representation of a 2D character matrix (chess matrix), to represent the status of the chess game, another matrix of gameobjects is applied to represent the chessman. If a proposed move (i_0, j_0, i_1, j_1) is valid, then chessman on $[i_0, j_0]$ can be moved to $[i_1, j_1]$. In this case, chessman $[i_0, j_0]$'s gameobject's position and rotation will be updated to the $\mathbf{p}[i_1, j_1]$ and $\mathbf{n}[i_1, j_1]$ respectively according to the parametric equation shown in the following section. The chessman $[i_1, j_1]$'s gameobject will be destroyed and chessman $[i_1, j_1]$'s gameobject will be assigned as chessman $[i_0, j_0]$'s gameobject and chessman $[i_0, j_0]$'s gameobject will be assigned to null. For example, using the same scenario shown in Figure 4, this time assumes it is a valid move, when a player moves the chessman bishop 'B' from $[i_0, j_0]$ to $[i_1, j_1]$, our approach modifies the chess matrix by setting chess $[i_1, j_1]$=$[i_0, j_0]$='B' and $[i_0, j_0]$=' '. Meanwhile, update the virtual scene to synchronize with the updated chess matrix, that is, update the chessman gameobjects by destroying $[i_1, j_1]$, translating and rotating $[i_0, j_0]$, assigning $[i_1, j_1]$=$[i_0, j_0]$ and $[i_0, j_0]$=null. So, as we can see, this matrix representation of chess is convenient for implementation.

Min-Max Algorithm. One of the challenging parts of this proposed work is to automatically determine the next move for computers using the game AI approach. The easiest way to do so is using the greedy algorithm. That is to find out which move can make the computer get the highest score according to the evaluation of the next status, But actually, this is a naive approach because it does not consider the opponent player's strategy. So here there are many advanced techniques that can be applied to solve this problem such as deep learning and reinforcement learning. But in this project, as this game AI part is not the main contribution of the work, we implemented a classical but robust game AI algorithm called Min-Max Algorithm which is widely used in game theory to deal with two-players game problems. The main idea of the Min-Max Algorithm is: a computer's best move needs to make the

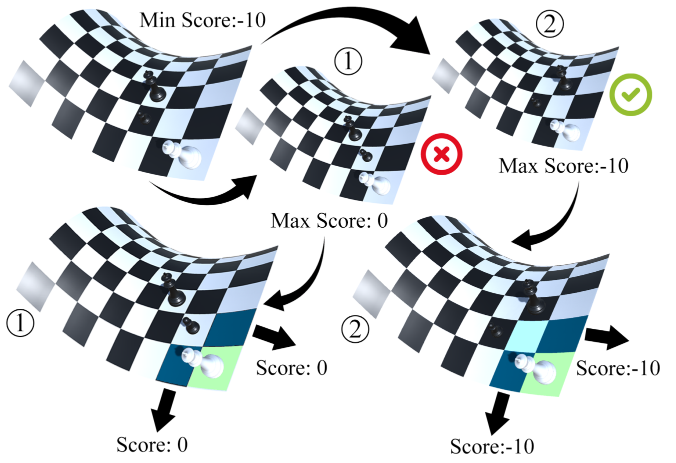


Fig. 5. Min-Max Algorithm. This figure shows an example of using the Min-Max Algorithm in the chess game AI. In this case, the black player has two options: (1) move the black pawn one step forward or (2) move the black king one step forward. Under option (1), either the white king moves one step forward or rightward, it will be not ”killed”. Then, the max score for the white player is 0; Under option (2), no matter how to move the white king, it will be ”killed”. Then, the max score for the white player is -10; Therefore, according to the Min-Max Algorithm, the black player will choose option (2) which results in the min value of the white player’s max score.

opponent player’s next best move worse. To describe this in another way, if a computer can get a maximal score of 10 and then it wins this game. On the opposite, if the opposing player gets a minimal score of -10 then will win. So the computer’s best move needs to make the opponent player’s next move get the max score. In a similar way, the opponent player’s best move needs to make the computer’s next move to get the min score. Namely, this is called the Min-Max algorithm.

Figure 5 shows the idea of the Min-Max Algorithm. Different possibilities are different valid moves or choices that can be made by the computer or its opponent player. Every time during the recursion, computer always find the the opponent player’s max score to return and the opponent player always find the computer’s min score to return. In order to speed up the search process, our approach adds the Alpha-Beta Cutoffs. The idea is that once it finds a max score for the current move, say alpha, in the later search, it needs to only consider the move which can help find a higher score than this alpha value, so all of the other cases will be cut off. The opposite is also true: It needs to only consider the move which can help find a lower score than the beta value, so all of the other cases will be cut off. For the reason that the search space for chess is too large, and therefore ordinary computers can not search too deeply. So here, a depth restriction is applied that if the current depth is larger than any threshold, it returns to the current scores. However, this restriction disables the computer to find the global optimum solution at the early stage.

Parametric Surface. One main contribution of this work is procedurally deploying chess games on parametric surfaces. Parametric surfaces are mathematically defined with given parametric equations $\mathbf{p}(u,v) = (x,y,z) \in \mathbb{R}^3$ that projects parametric space $(u,v) \in \mathbb{R}^2$ into world space $(x,y,z) \in \mathbb{R}^3$ so as to generate surfaces and shapes as 3D models. In our work, the plane surface of chessboard are transformed into parametric surfaces using parametric equations that takes the input of the coordinates of the squares, namely, $[i,j]$, which denotes the i^{th} row and the j^{th} column on the chessboard. Then, the vertices of the triangle mesh for chessboard square $[i,j]$ are calculated through following point set $\mathbf{S}[i,j]$:

$$\mathbf{S}[i,j] = \bigcup_{m=0}^{k-1} \bigcup_{n=0}^{k-1} \mathbf{p}(i + \frac{m}{k-1}, j + \frac{n}{k-1}),$$

where coordinates i and j are integers range from 0 to 7. Empirically, we set surface precision $k = 5$. Chessman’s position on square $[i,j]$ is $\mathbf{p}[i,j] = \mathbf{p}(i+0.5, j+0.5)$ and the chessman’s rotation on square $[i,j]$ align with the surface’s normal directions $\mathbf{n}[i,j]$ which is calculated as:

$$\mathbf{n}[i,j] = \frac{(\mathbf{p}(i+\Delta,j) - \mathbf{p}(i,j)) \times (\mathbf{p}(i,j+\Delta) - \mathbf{p}(i,j))}{|(\mathbf{p}(i+\Delta,j) - \mathbf{p}(i,j)) \times (\mathbf{p}(i,j+\Delta) - \mathbf{p}(i,j))|},$$

where the numerical partial differentiation step $\Delta = 0.001$. In the end, give the up vector $\mathbf{u} = (0,1,0)$, the chessman on square $[i,j]$ will rotate along the axis $\mathbf{a}[i,j] = \mathbf{n}[i,j] \times \mathbf{u}$ with the angle of $\alpha[i,j] = \cos^{-1}(\mathbf{n}[i,j] \cdot \mathbf{u})$.

V. Experiment Result

In order to validate the effectiveness and correctness of our approach, we conduct a series of experiments to deploy chess games on different parametric surfaces. Figure 6 shows the results that deploy chess game on heightmap parametric surfaces including (a) plane surface, (b) water wave surface, (c) sin-cos surface, (d) parabola surface, (e) Gaussian surface, and (f) helmet surface. Mathematically, let $w = 7$ denote the square size, then these heightmap parametric surfaces can be represented by heightmap function $h(u,v)$ such that $\mathbf{p}(u,v) = (wu, h(u,v)/w, wv)$. Let r denotes the distance from (u,v) to the center $(4,4)$ which can be calculated as:

$$r = \sqrt{(u-4)^2 + (v-4)^2}$$

In Figure 6 (a), heightmap function of plane surface is:

$$h(u,v) = 0$$

In Figure 6 (b), heightmap function of water wave surface is:

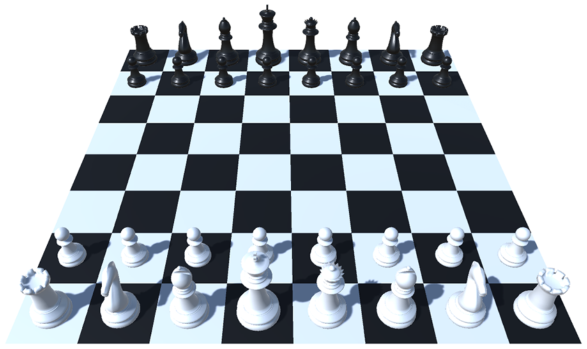
$$h(u,v) = 3\cos(2\pi r/3)/\exp(r^2/4)$$

In Figure 6 (c), heightmap function of sin-cos surface is:

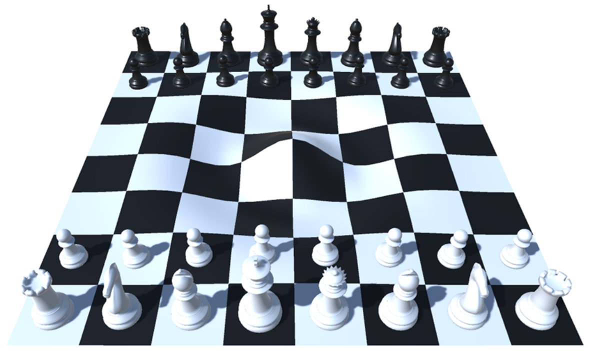
$$h(u,v) = 4\sin(u\pi/4)\cos(v\pi/4)$$

In Figure 6 (d), heightmap function of parabola surface is:

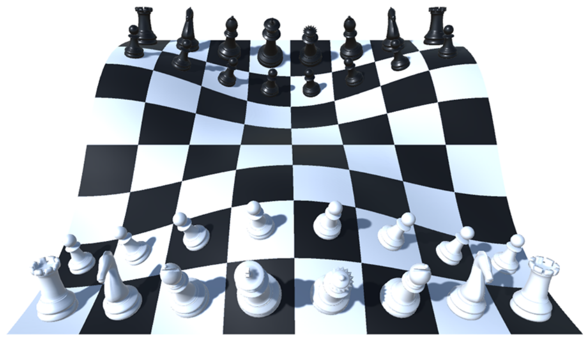
$$h(u,v) = r^2/2 - 10$$



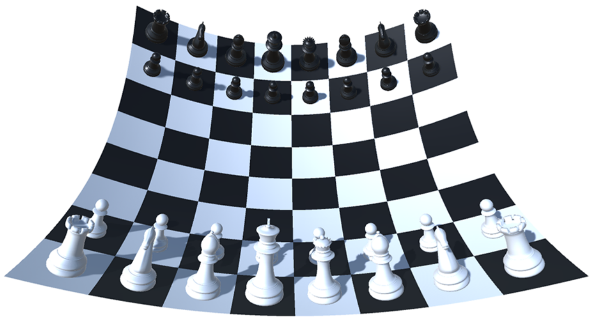
(a) plane surface



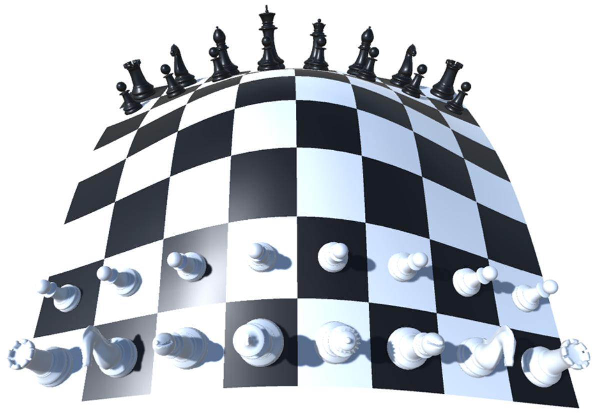
(b) water wave surface



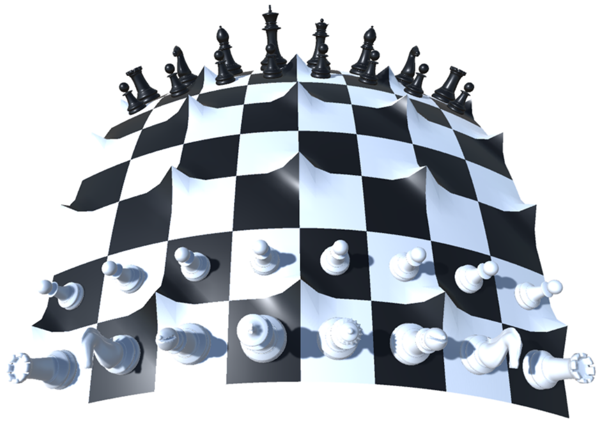
(c) sin-cos surface



(d) parabola surface



(e) Gaussian surface



(f) helmet surface

Fig. 6. Experiment Results. This figure shows results that deploy chess game on six parametric surfaces including: (a) plane surface, (b) water wave surface, (c) sin-cos surface, (d) parabola surface, (e) Gaussian surface, and (f) helmet surface.

In Figure 6 (e), heightmap function of Gaussian surface is:

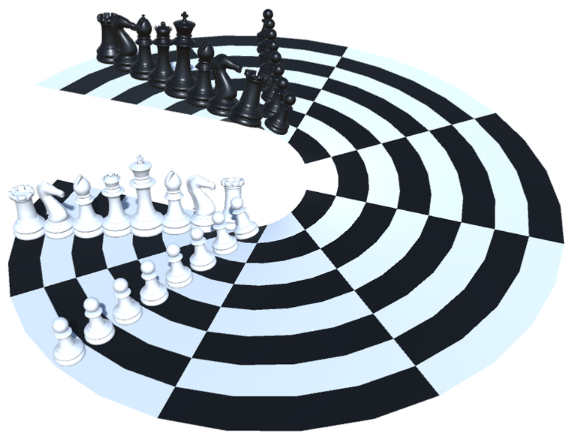
$$h(u, v) = 20\exp(r^2/16) - 10$$

In Figure 6 (f), heightmap function of helmet surface is:

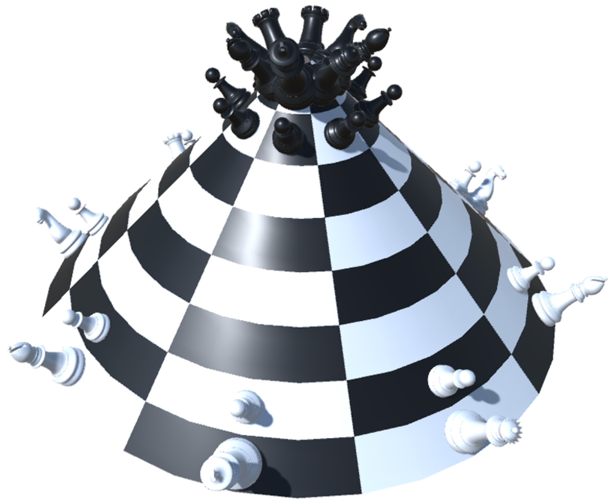
$$h(u, v) = 3\sin^{16}((u-4)\pi/2)\cos^{16}((v-4)\pi/2) - r^2/2 + 6$$

As we can tell from the result, the chessboard's geometry is correctly generated with these heightmap parametric surfaces and the chessmen are accurately aligned at the center of each square and the rotation is along the normal direction. This proves the correctness of our approach in deploying chess games on arbitrary smooth heightmap parametric surfaces. As another application of heightmap parametric surface, digital terrain with smooth heightmap data can also be taken as the input of our approach.

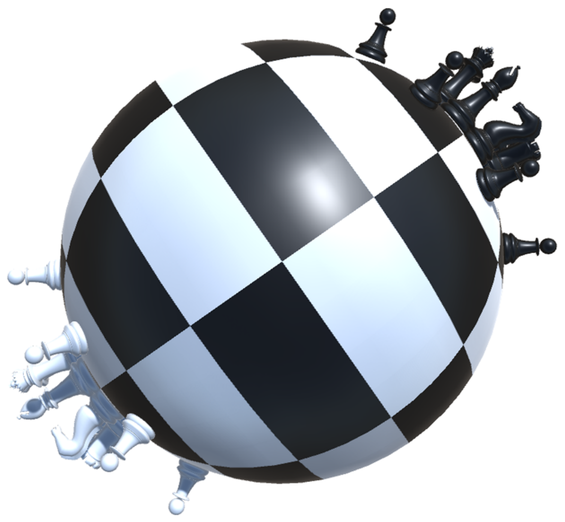
However, the heightmap parametric surfaces that are represented by scalar height field equations (so-called heightmap equations) are a special branch of the parametric surface



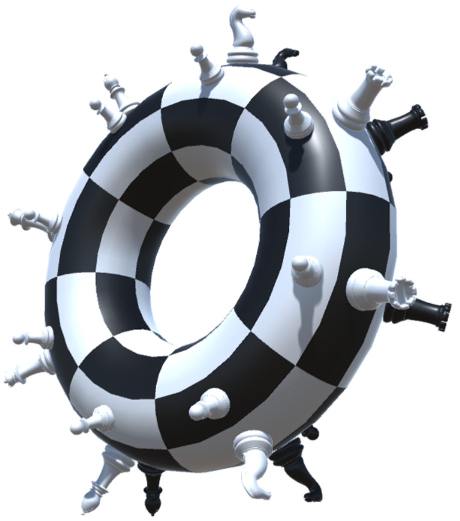
(a) helicoid surface



(b) cone surface



(c) sphere surface



(d) torus surface

Fig. 7. Experiment Results (Cont.). This figure shows results that deploy chess game on other four different types of parametric surfaces including (a) helicoid surface, (b) cone surface, (c) sphere surface, and (d) torus surface which are not heightmap-based.

family. More general forms of parametric surfaces are not only projecting from 2D parameter space to 3D world space by only adding the height information, but rather, by mapping any 2D points in parameter space to an arbitrary 3D point in the world space. As a sequence, the result parametric surface can be closed surfaces such as spheres and torus. Without losing generality, we test our approach in deploying chess games on generalized parametric surfaces that are not based on heightmap equations. Figure 7 shows the results that deploy chess game on other four different types of parametric surfaces including (a) helicoid surface, (b) cone surface, (c) sphere surface, and (d) torus surface. Let square size $w = 7$. In Figure 6 (a), the parametric equation of helicoid surface is:

$$\mathbf{p}(u,v) = \frac{w}{2}\left(-(v+1)\cos(u\frac{\pi}{4}), -\frac{u-4}{2}, (v+1)\sin(u\frac{\pi}{4})\right)$$

In Figure 6 (b), the parametric equation of cone surface is:

$$\mathbf{p}(u,v) = \frac{w}{2}\left((u+\frac{1}{4})\cos(v\frac{\pi}{4}), u, (u+\frac{1}{4})\sin(v\frac{\pi}{4})\right)$$

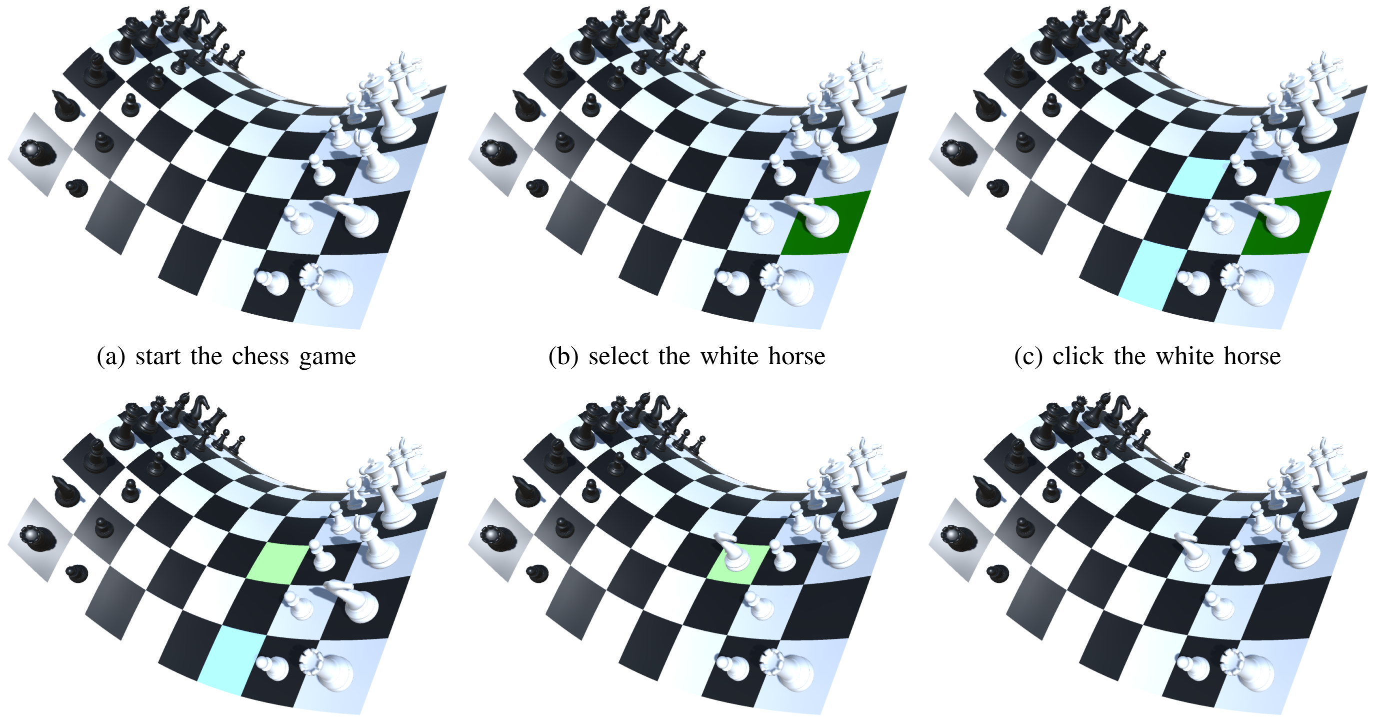
In Figure 6 (c), the parametric equation of sphere surface is:

$$\mathbf{p}(u,v) = 3w\left(\cos(v\frac{\pi}{4})\sin(u\frac{\pi}{8}), \sin(v\frac{\pi}{4})\sin(u\frac{\pi}{8}), \cos(u\frac{\pi}{8})\right)$$

In Figure 6 (d), the parametric equation of torus surface is:

$$\mathbf{p}(u,v) = \frac{w}{2}\begin{pmatrix} (5+2\cos(u\pi/4))\cos(v\pi/4) \\ (5+2\cos(u\pi/4))\sin(v\pi/4) \\ 2\sin(u\pi/4) \end{pmatrix}$$

As shown in these results, the chessboard's geometry is correctly generated with these generalized parametric surfaces as well and the chessmen are accurately aligned at the center of each square and the rotation is along the normal direction. These results prove the correctness of our approach in deploying chess games on generalized parametric surfaces that are not based on heightmap equations.


(a) start the chess game (b) select the white horse (c) click the white horse

(d) select the hint square (e) click the hint square (f) move the black pawn by AI

Fig. 8. Interaction Experiment. This figure shows the experiment result that tests a user's interaction with our chess game interface. In this case, the chess is deployed on a saddle surface when the game starts (a). Then, subfigures (b)-(f) show the corresponding effects when the user's mouse (or VR controller) triggers the following events including (b) select the white horse, (c) click the white horse, (d) select the hint square, (e) click the hint square, and (f) move the black pawn via game AI.

In order to test the dynamic feature of our approach that the user can interact with the computer and move the chess correctly through our proposed interface, we conduct the experiment to test the user interactions parts and game AI aspects of our interface. Figure 8 shows the experiment result that tests a user's interaction with the computer using our proposed chess game interface. In this case, the chess is deployed on a heightmap-based parametric surface: saddle surface, whose heightmap equation is:

$$h(u,v) = \frac{1}{10}\left((u-4)^2 - (v-4)^2\right)$$

When the game starts, the player selects the white horse when the mouse (or VR controller) points at it. As shown in Figure 8(b), the square under the white horse is highlighted as dark green. As shown in Figure 8(c), when the player clicks that dark green square, hint squares in light blue pop up to show the valid moves of this white horse. As shown in Figure 8(d), when the mouse (or VR controller) points at a hint square, that hint square is highlighted as light green. As shown in Figure 8(e), when the player clicks the hint square, the white horse moves to that hint square correspondingly. As shown in Figure 8(f), once the player finished each valid move, the game AI will move the black chessman using the Min-Max algorithm which moves the black pawn one step forward.

Implementation of the selection mechanism is done by adding two components to the square's GameObject which are Rigidbody and SphereCollider. The radius size of SphereCollider is half of the square size w and the position of the SphereCollider is exactly the same as the chessman's position on that square which is calculated through the parametric equations. Squares' color changing is implemented by switching the predefined materials during the gameplay. For example, by default, the square's colors are black or white. After being selected, they will become dark green or light green. If they are hints for valid moves, their colors will become dark blue or light blue. During the gameplay, color changes can help players distinguish which is the current status of chess squares.

VI. User Study

SurfChessVR is implemented on Unity3D Editor and executes on the Unity3D game engine. Users can play SurfChessVR on a desktop through mouse interactions. In the meantime, with the SteamVR plugin imported into our project, users can also play SurfChessVR on VR devices such as Oculus Quests. We have conducted a preliminary user study to test our proposed chess gaming interface, SurfChessVR, in virtual reality. As shown in Figure9, the user plays chess on a torus surface against the game AI in virtual reality through the Oculus Quest 2 VR headset and VR controllers. When the user clicks on the hint squares with her VR controller's grab pinch button, that chessman moves to that hint square and "eats" the black pawn there. During this user study, we invite this user to play SurfChessVR chess games with eleven different chessboard shapes including plane surface, water wave surface, sin-cos surface, parabola surface, Gaussian surface, helmet surface, saddle surface, helicoid surface, cone surface, sphere surface and torus surface which are shown in the experiment results. Full video recording of the user study can be accessed through this link [26]. After the study, the user shows a high evaluation of SurfChessVR and enjoys the interactions of this interface. She also believes the chessboards generated from parametric surfaces look very special and believe they can make chess games gain more popularity among chess fans.

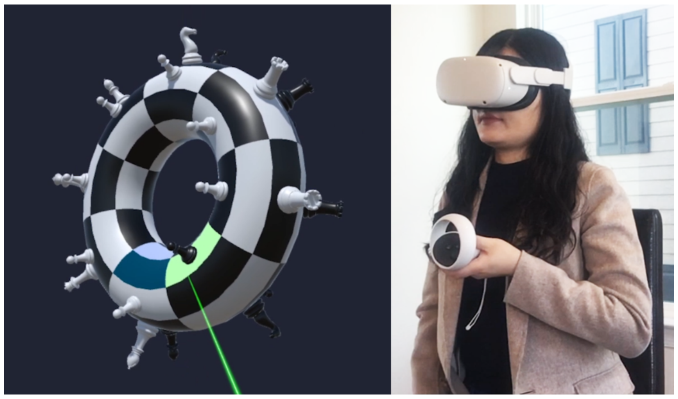


Fig. 9. User Study. This figure shows the user's gameplay experience with SurfChessVR. In this example, the user plays chess on a torus surface against the game AI in virtual reality through the Oculus Quest 2 VR headset and VR controllers.

VII. Conclusion

In this paper, we propose SurfChessVR, a novel gaming interface that deploys chess games on parametric surfaces in Virtual Reality (VR). This is the first research work that applies parametric modeling techniques to chessboard synthesis and chess game procedural content generation. More specifically, our technical approach can automatically generate virtual chessboards with different shapes of parametric surfaces and automatically place chessmen on that parametric surface with realistic alignments. We also validate the effectiveness and correctness of our approach through a group of experiments and user studies on virtual reality devices. In future work, we will explore our interface for generating more complex parametric surfaces. Especially, we will explore the possibility to extend our approach to generating rough surfaces which have abrupt changes in normal directions. Our proposed interface can also be applied to user studies that evaluate the aesthetics and the intractability of chessboards with different shapes.

References

[1] Chessvariants, "Spherical chess," https://www.chessvariants.com/boardrules.dir/spherical.html, Sep 1996.

[2] M. Davis, "Spherical chess," http://www.endprod.com/chess/sphere.htm, 1976.

[3] ——, "Round chess," http://www.endprod.com/chess/round.htm, 1976.

[4] Chessvariants, "Cylindrical chess," https://www.chessvariants.com/boardrules.dir/cylindrical.html, Sep 1996.

[5] B. Bontchev, "A mobile chess game," 2008.

[6] L.-H. Chen, C.-J. Yu, and S.-C. Hsu, "A remote chinese chess game using mobile phone augmented reality," in *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, 2008, pp. 284–287.

[7] G. Kaur, A. K. Yadav, and V. Anand, "Design and implementation of artificially intelligent microcontroller based chess opponent," in *Proceedings of the World Congress on Engineering*, vol. 1, 2010.

[8] X. Li and K.-S. Hong, "Korean chess game implementation by hand gesture recognition using stereo camera," in *2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT)*, vol. 2. IEEE, 2012, pp. 741–744.

[9] K.-L. Su, B.-Y. Li, J.-H. Guo, and K.-H. Hsia, "Implementation of the chess game artificial intelligent using mobile robots," in *2014 Joint 7th International Conference on Soft Computing and Intelligent Systems (SCIS) and 15th International Symposium on Advanced Intelligent Systems (ISIS)*. IEEE, 2014, pp. 169–174.

[10] A. R. Mendes, A. M. Mehta, and B. H. Gohil, "Implementation of the automatic and interactive chess board," *ISOR Jurnal of Electrical and Electronics Engineering*, vol. 9, pp. 1–4, 2014.

[11] F. A. T. Al-Saedi and A. H. Mohammed, "Design and implementation of chess-playing robotic system," *International Journal of Science, Engineering and Computer Technology*, vol. 5, no. 5, p. 90, 2015.

[12] M. Peiravi, "The design and implementation of an adaptive chess game," 2015.

[13] S. Z. M. Muji, M. H. A. Wahab, R. Ambar, and W. K. Loo, "Design and implementation of electronic chess set," in *2016 International Conference on Advances in Electrical, Electronic and Systems Engineering (ICAEES)*. IEEE, 2016, pp. 451–456.

[14] G. Wu and J. Tao, "Chinese chess algorithm design and implementation in the computer games," in *2016 35th Chinese Control Conference (CCC)*. IEEE, 2016, pp. 10 380–10 384.

[15] S. Li, X. Yuan, and K. Cao, "Design and implementation of personalized interface of chinese army chess," in *2016 Chinese Control and Decision Conference (CCDC)*. IEEE, 2016, pp. 4271–4274.

[16] Y.-K. Kim, Y.-S. Yoon, T.-G. Oh, Y.-H. HwangBo, and J.-H. Hwang, "Real-time vr strategy chess game using motion recognition," *Journal of Digital Contents Society*, vol. 18, no. 1, pp. 1–7, 2017.

[17] D. Bhutani, Y. Ali, and P. Gupta, "Design and implementation of a wireless remote chess playing physical platform," *International Journal of Engineering Research and Technology*, vol. 6, no. 09, 2017.

[18] G. O. Larregay, F. L. Pinna Gonzalez, L. O. Avila, and O. D. Morán, "Design and implementation of a computer vision system for an autonomous chess-playing robot," 2018.

[19] C. S. Yusof, T. S. Low, A. W. Ismail, and M. S. Sunar, "Collaborative augmented reality for chess game in handheld devices," in *2019 IEEE Conference on Graphics and Media (GAME)*. IEEE, 2019, pp. 32–37.

[20] Y. Zhong, "Object-oriented implementation of chess game in c++," in *Journal of Physics: Conference Series*, vol. 1195, no. 1. IOP Publishing, 2019, p. 012013.

[21] L. Shi, S. Li, D. Lei, and J. Bo, "Design and implementation of a general chess game system client based on electron framework," in *2019 IEEE International Conferences on Ubiquitous Computing & Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS)*. IEEE, 2019, pp. 696–701.

[22] J. Wang, X. Wu, T. Qian, H. Luo, and C. Hu, "Design and implementation of chinese chess based on manipulator," in *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. IEEE, 2019, pp. 1687–1690.

[23] M. Li and W. Huang, "Research and implementation of chinese chess game algorithm based on reinforcement learning," in *2020 5th International Conference on Control, Robotics and Cybernetics (CRC)*. IEEE, 2020, pp. 81–86.

[24] P. Kołosowski, A. Wolniakowski, and K. Miatliuk, "Collaborative robot system for playing chess," in *2020 International Conference Mechatronic Systems and Materials (MSM)*. IEEE, 2020, pp. 1–6.

[25] J. Zhang and H. Yin, "Design and implementation of intelligent chinese chess system device," in *2020 International Conference on Culture-oriented Science & Technology (ICCST)*. IEEE, 2020, pp. 558–563.

[26] A. Author(s), "Surfchessvr: Deploying chess game on parametric surface in virtual reality (video)," https://youtu.be/ee40Q4ibsf8, 2023.