

גרפיקה ממוחשבת  
89-663-02

מקליד: עידן אריה  
מתרגל: ד"ר אורן קאפח  
תאריך: 2019-05-15

יש לנו שחקן במיקום  $P(x, y, z)$  והמצלמה היא בזווית  $\alpha$  (בין ציר ה- $X$  לציר ה- $Z$ ). נשתמש בפונקציה `gluLookAt` כדי לגרום למצלמה להסתכל לכיוון הנכון. נבחר נקודה במרחק 1 (לשם הפשטות) בכיוון הרצוי:

```
gluLookAt(Px, Py, Pz, Px + cos(alpha), Py, Pz + sin(alpha), 0, 1, 0);
```

יש 9 פרמטרים שמחולקים ל-3 ווקטורים:

- הנקודה ממנה מסתכלים.
- הנקודה עליה מסתכלים.
- ווקטור ה"למעלה".

כדי לקדם את השחקן, פשוט צריך לשנות את  $P$ :

```
case 'W': // go forward
    Px += step * cos(alpha);
    Pz += step * sin(alpha);
    break;
case 'S': // go backward
    Px -= step * cos(alpha);
    Pz -= step * sin(alpha);
    break;
```

כדי לסובב את השחקן, צריך רק לשנות את הזווית  $\alpha$ :

```
case LEFT_ARROW: // turn left
    alpha += alphaStep;
    break;
case RIGHT_ARROW: // turn right
    alpha -= alphaStep;
    break;
```

כדי לזוז שמאלה או ימינה, צריך לזוז כמו קדימה/אחורה - רק בזווית  $\alpha + 90^\circ$ :

```
case 'A': // sidestep left
    Px += step * cos(alpha + 90);
    Pz += step * sin(alpha + 90);
    break;
case 'D': // sidestep right
    Px -= step * cos(alpha + 90);
    Pz -= step * sin(alpha + 90);
    break;
```

ומה אם אנחנו רוצים גם להסתכל למעלה ולמטה? אנחנו צריכים עוד זווית -  $\beta$  - בין המישור  $XZ$  לציר ה- $Y$ .  
מתוך  $\alpha, \beta$ , אנחנו יכולים לחשב את ווקטור הכיוון  $D$ :

$$D : \quad \begin{aligned} y &= \sin \beta \\ x &= \cos \beta \cdot \cos \alpha \\ z &= \cos \beta \cdot \sin \alpha \end{aligned}$$

אבל עכשיו ווקטור ה"למעלה" שלנו הוא כבר לא  $[0 \ 1 \ 0]$  פשוט. כדי למצוא אותו, נעשה אותו חישוב כמו  $D$  - רק עם  $\beta + 90^\circ$ :

$$UP : \quad \begin{aligned} y &= \sin (\beta + 90^\circ) \\ x &= \cos (\beta + 90^\circ) \cdot \cos \alpha \\ z &= \cos (\beta + 90^\circ) \cdot \sin \alpha \end{aligned}$$

כשנרצה לזוז קדימה/אחורה, נעשה  $P \pm = UP \cdot \text{step}$ . ובשביל לזוז שמאלה/ימינה, ניצור ווקטור חדש עם הזוויות  $\beta, \alpha + 90^\circ$ .

## בעיות אפשריות

כאשר  $-90^\circ < \beta < 90^\circ$  זה עובד טוב - אבל כאשר  $\beta$  עוברת את  $\pm 90^\circ$  הכיוונים שמאלה וימינה כבר לא כל כך מוגדרים.

## פתרון

במקום להחזיק את הכיוון עם  $\alpha, \beta$ , נגדיר מערכת צירים של השחקן. זה גם יתן לנו את ווקטורים הקדימה/למעלה/הצידה, וגם נוכל לסובב את מערכת הצירים בכל ציר שנרצה - כולל גלגול על הצד (כמו במשחקי מטוסים/חלליות). כדי לסובב את מערכת הצירים סביב ציר ה- $Y$ :

$$\begin{aligned} \hat{x}_n &= \hat{x} \cos \alpha - \hat{z} \sin \alpha \\ \hat{y}_n &= \hat{y} \\ \hat{z}_n &= \hat{z} \cos \alpha + \hat{x} \sin \alpha \end{aligned}$$

וסביב צירים אחרים בצורה דומה.

## טקסטורה

כדי לטעון טקסטורה נשתמש בפונקציה `TextureIO.newTexture` שמקבלת קובץ שממנו תטען את הטקסטורה ופרמטר בוליאני שקובע אם לייצר `mip-maps`. צריך לשים לב שמימדי התמונה הן חזקות של 2. כדי להדביק את הטקסטורה על האובייקט, נרצה להצמיד נקודות של הטקסטורה (במערכת קואורדינטות בין 0 ל 1 על כל ציר) לקודקודים של האובייקט. מבצעים זאת על ידי קריאה ל `gl.glTexCoord2f` עם הקואורדינטות של הטקסטורה לפני שמגדירים כל אובייקט. אבל לפני שבכלל מתחילים את ציור האובייקט צריך לעשות `.bind()` לטקסטורה, כדי ש `OpenGL` ידע להשתמש בה. יש כמה פרמטרים שאפשר להגדיר לטקסטורה:

- `Wrapping` - מה קורה אם חורגים מגבולות הטקסטורה  $[0, 1]$ ?
  - `GL.GL_CLAMP` - שימשיך את הטקסטורה לפי הפיקסל האחרון (צבע אחיד)
  - `GL.GL_REPEAT` "ישכפל" את הטקסטורה
- `Filtering` - מה קורה אם הטקסטורה לא בדיוק בגודל שצריך לצייר אותה (יקרה כמעט תמיד, כי זה משתנה לפי המרחק)
  - `GL_NEAREST` - בוחר את הפיקסל הכי קרוב. יותר מהיר אבל פחות יפה
  - `GL_LINEAR` - מחשב מהפיקסלים ליד את הצבע הנכון. יותר איטי
- `MipMaps` - יוצר כמה עותקים של הטקסטורה ברזולוציות שונות, כדי לייעל את `Filtering`.