**פרויקט סיום קורס שפת תכנון חומרה באמצעות וורילוג**

**נושא הפרויקט: Gaussian Filter**

**תיאור הפרויקט:**

**רקע תיאורטי:**

טשטוש גאוסיאני הינו טשטוש המשתמש בעקומת גאוס כבסיס ההתמרה. פילטר זה גורם לקווי המתאר בתמונה להיות קלים יותר לזיהוי. הוא גורם לכך שהערכים במרכז מקבלים ערכים גבוהים יותר לפי עקומת גאוס. הפילטר פועל לפי הפונקציה הבאה:

כאשר (x,y) הן הקואורדינטות (x ו y במרכז שווים לאפס)

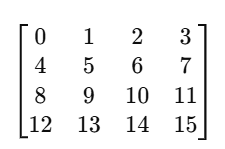
זה הסטיית תקן.

הגדלת סיגמא תיצור טשטוש גדול יותר.

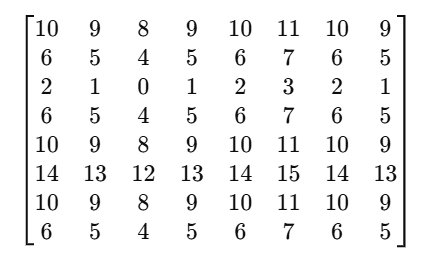
הוספת הפילטר לתמונה פועל בדרך הבאה: לקיחת חלון בתמונה המקורית בגודל תואם לפילטר. ביצוע קונבולוציה, כלומר הכפלת הערכים של החלון והפילטר במקומות התואמים וסכימה. הסקלר שמקבלים הוא הערך החדש של התא במטריצה שנמצא במרכז של אותו חלון שנלקח מהתמונה. לאחר שמעדכנים אותו מזיזים את החלון צעד אחד קדימה וחוזרים על אותה פעולה עד שמסיימים לרוץ על כל התמונה.

חשוב לציין שבשימוש בפילטרים שונים כמו פילטר גאוסי, קיימות דרכים שונות של טיפול בקצוות. למשל ניתן לרפד את הקצוות שחורגות מהתמונה באפסים, בערכים של הקצוות של התמונה או לא לעדכן אותם בכלל. פעולה נפוצה אחרת היא לעשות mirroring כך שהערכים שחורגים מגבולות התמונה מקבלים ערכים דומים לערכים שמקבלים אם מתקדמים בכיוון השני.

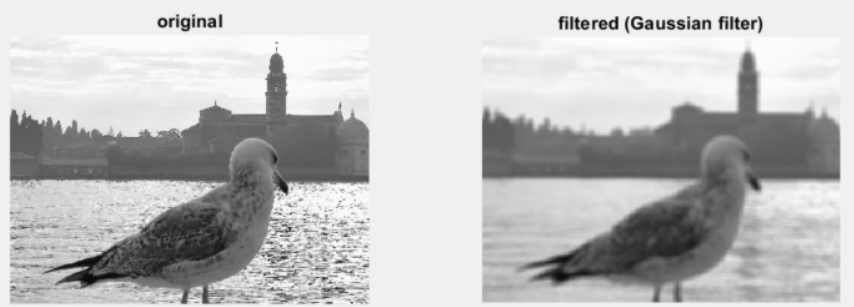
לדוגמה המטריצה:



תקבל את הערכים:



בתמונה הבאה ניתן לראות דוגמה לשימוש בפילטר.

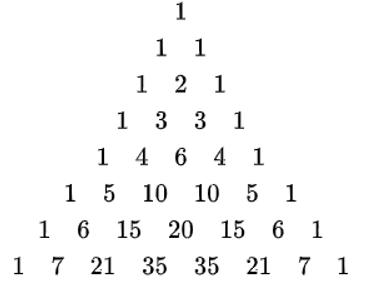


**תיאור הפרויקט:**

**רקע תיאורטי - המשך:**

**משולש פסקל:**

משולש פסקל הוא סידור של מספרים בצורת משולש, הנבנה באופן הבא: הקודקוד העליון של משולש זה מכיל את המספר 1, וכל מספר במשולש מהווה את סכום שני המספרים שנמצאים מעליו (המספרים שנמצאים על שוקי המשולש הם כולם 1).



משולש פסקל ידוע בעיקר בגלל השימוש המגוון שלו והתכונות הייחודיות הרבות שיש לו. במקרה שלנו התכונה השימושית עבורנו היא ההתפלגות הנורמלית שנוצרת בכל שורה. בעזרת תכונה זו ניתן לפשט את חישוב הפילטר הקוד, כך שבמקום שנשתמש במשוואת גאוס שקשה ליישום בשפות כמו וורילוג, ניתן להשתמש בנוסחה לחישוב ערך האיברים (C) הנמצאים במקום ה k שבשורה ה n (כש n שווה לגודל הפילטר פחות אחד).

כך ניצור שני וקטורים זהים שמתפלגים נורמלית ובמטריצה של הפילטר נשים בשורה הראשונה את הוקטור כשכל איבריו מוכפלים באיבר הראשון של הוקטור השני, בשורה השנייה נשים שוב את אותו הוקטור אבל כעת הוא יהיה מוכפל באיבר השני של הוקטור השני וכן הלאה. כך נקבל מטריצה עם התפלגות נורמלית. כעת מה שנותר זה רק לסכום את כל הערכים במטריצה ולחלק אותם בסכום, כך הערכים יהיו מנורמלים.

**פרויקט סיום** - **קוד המערכת:**

**module** GaussFilter **#(parameter** rows **=** 4**,** cols **=** 4**,** ksize **=** 3**,** data\_width **=** 8**)** **(**

**input** **wire** clk**,**

**input** **wire** rst**,**

**input** **wire** start**,**

**input** **wire** **[**data\_width**\***rows**\***cols**-**1**:**0**]** image\_in**,**

**output** **reg** **[**data\_width**\***rows**\***cols**-**1**:**0**]** image\_out**,**

**output** **reg** image\_out\_valid**,**

**output** **reg** done**,**

**input** **wire** image\_ready

**);**

**reg** **[**data\_width**-**1**:**0**]** memory **[**0**:**rows**\***cols**-**1**];** // To store memory

**reg** **[**7**:**0**]** SigmaMatrix **[**0**:**ksize**\***ksize**-**1**];** // Kernel

**integer** i**,** j**,** k**,** l**,** y**,** z**,** ii**,** n**,** nn**;**

**reg** **[**15**:**0**]** sum\_g**;** // Accumulator for convolution

**reg** **[**7**:**0**]** result **[**0**:**rows**\***cols**-**1**];** // Result of convolution

**reg** **[**7**:**0**]** ColumnVector **[**0**:**ksize**-**1**];** // Column vector for kernel calculation

**reg** **[**7**:**0**]** RowVector **[**0**:**ksize**-**1**];** // Row vector for kernel calculation

**reg** **[**3**:**0**]** state**;** // state machine

**integer** pixel\_idx**;** // Pixel index

// FSM States

**localparam** IDLE **=** 0**,** CALC\_KERNEL **=** 1**,** LOAD\_IMAGE **=** 2**,** CONVOLVE **=** 3**,** OUTPUT\_RESULT **=** 4**,** DONE **=** 5**;**

**function** **integer** factorial**;**

**input** **integer** num**;**

**integer** fact**,** i**;**

**begin**

fact **=** 1**;**

**for** **(**i **=** 1**;** i **<=** num**;** i **=** i **+** 1**)**

fact **=** fact **\*** i**;**

factorial **=** fact**;**

**end**

**endfunction**

**always** **@(posedge** clk **or** **posedge** rst**)** **begin**

**if** **(**rst**)** **begin**

state **<=** IDLE**;**

image\_out **<=** 0**;**

done **<=** 0**;**

pixel\_idx **<=** 0**;**

sum\_g **<=** 0**;**

k **<=** 0**;**

l **<=** 0**;**

image\_out\_valid **<=** 1'b0**;**

**end** **else** **begin**

**case** **(**state**)**

IDLE**:** **begin**

**if** **(**start**)** **begin**

state **<=** CALC\_KERNEL**;**

**end**

**end**

// Compute Kernel

CALC\_KERNEL**:** **begin**

// Calculate the kernel values

**for** **(**i **=** 0**;** i **<** ksize**;** i **=** i **+** 1**)** **begin**

ColumnVector**[**i**]** **=** factorial**(**ksize**-**1**)** **/** **(**factorial**(**i**)** **\*** factorial**(**ksize**-**1**-**i**));**

RowVector**[**i**]** **=** factorial**(**ksize**-**1**)** **/** **(**factorial**(**i**)** **\*** factorial**(**ksize**-**1**-**i**));**

**end**

// Compute the 2D Gaussian kernel

**for** **(**ii **=** 0**;** ii **<** ksize**;** ii **=** ii **+** 1**)** **begin**

**for** **(**j **=** 0**;** j **<** ksize**;** j **=** j **+** 1**)** **begin**

SigmaMatrix**[**ii **\*** ksize **+** j**]** **=** ColumnVector**[**ii**]** **\*** RowVector**[**j**];**

**end**

**end**

// Normalize the kernel

sum\_g **=** 0**;**

**for** **(**n **=** 0**;** n **<** ksize**\***ksize**;** n **=** n **+** 1**)** **begin**

sum\_g **=** sum\_g **+** SigmaMatrix**[**n**];**

**end**

**for** **(**nn **=** 0**;** nn **<** ksize**\***ksize**;** nn **=** nn **+** 1**)** **begin**

SigmaMatrix**[**nn**]** **=** **(**SigmaMatrix**[**nn**]\***256**)** **/** sum\_g**;**

**end**

**if** **(**image\_ready**)**

state **<=** LOAD\_IMAGE**;**

**end**

// Load input image into memory

LOAD\_IMAGE**:** **begin**

memory**[**pixel\_idx**]** **<=** image\_in**[**pixel\_idx**\***data\_width **+:** data\_width**];** // size: data\_width. starts from pixel\_idx\*data\_width

pixel\_idx **<=** pixel\_idx **+** 1**;**

**if** **(**pixel\_idx **==** rows **\*** cols **-** 1**)** **begin**

pixel\_idx **<=** 0**;**

state **<=** CONVOLVE**;**

**end**

**end**

// Convolution

CONVOLVE**:** **begin**

i **=** pixel\_idx **/** cols**;** // Current row

j **=** pixel\_idx **%** cols**;** // Current column

**if** **(**k **<** ksize**)** **begin** //row index inside the window in the image

**if** **(**l **<** ksize**)** **begin** //Column index inside the window in the image

y **=** i **+** k **-** ksize**/**2**;** // Row offset. [Row in image] + [Row in window]- ksize/2 (index in the middle is zero)

z **=** j **+** l **-** ksize**/**2**;** // Column offset.

**if** **(**y **>=** 0 **&&** y **<** rows **&&** z **>=** 0 **&&** z **<** cols**)** **begin** //condition for boundary check

sum\_g **<=** sum\_g **+** **(**memory**[**y **\*** cols **+** z**]** **\*** SigmaMatrix**[**k **\*** ksize **+** l**]);**

**end** **else** **begin**

// Handle edge cases by mirroring

y **=** **(**y **<** 0**)** **?** **-**y **:** **(**y **>=** rows**)** **?** 2**\***rows **-** y **-** 1 **:** y**;**

z **=** **(**z **<** 0**)** **?** **-**z **:** **(**z **>=** cols**)** **?** 2**\***cols **-** z **-** 1 **:** z**;**

sum\_g **<=** sum\_g **+** **(**memory**[**y **\*** cols **+** z**]** **\*** SigmaMatrix**[**k **\*** ksize **+** l**]);**

**end**

l **<=** l **+** 1**;**

**end** **else** **begin**

l **<=** 0**;**

k **<=** k **+** 1**;**

**end**

**end** **else** **begin**

result**[**pixel\_idx**]** **<=** sum\_g**>>**8**;** // Store result

pixel\_idx **<=** pixel\_idx **+** 1**;** // Move to next pixel

k **<=** 0**;**

l **<=** 0**;**

sum\_g **<=** 0**;**

**if** **(**pixel\_idx **==** rows **\*** cols **-** 1**)** **begin**

state **<=** OUTPUT\_RESULT**;**

pixel\_idx **<=** 0**;**

**end**

**end**

**end**

// Store convolution results

OUTPUT\_RESULT**:** **begin**

image\_out**[**pixel\_idx**\***data\_width **+:** data\_width**]** **<=** result**[**pixel\_idx**];**

pixel\_idx **<=** pixel\_idx **+** 1**;**

image\_out\_valid **<=** 1'b1**;**

**if** **(**pixel\_idx **==** rows **\*** cols **-** 1**)** **begin**

state **<=** DONE**;**

**end**

**end**

// Indicate completion

DONE**:** **begin**

done **<=** 1**;**

state **<=** IDLE**;**

image\_out\_valid **<=** 1'b0**;**

**end**

**default:** state **<=** IDLE**;**

**endcase**

**end**

**end**

**endmodule**

**פרויקט סיום** - **הסבר על קוד המערכת:**

הקוד מממש מערכת שעושה טשטוש גאוסי לתמונה בגודל 4x4 עם פילטר בגודל 3x3. את הגדלים ניתן לשנות בקלות ע"י שינוי הפרמטרים בתחילת הקוד.

המערכת כוללת שעון פעיל בעלייה, reset אסינכרוני פעיל בעלייה. בנוסף כניסה start שגורמת להתחלת הפעולה, image\_in שזה וקטור בו נמצאים ערכי התמונה בצורה חד ממדית ועוד כניסה image\_ready אותה מפעילים לאחר טעינת התמונה על מנת לבצע את הטשטוש עצמו.

במוצא המערכת יש image\_out, בו מכניסים את הערכים של התוצאה. image\_out\_valid כאשר מקבלים את המוצא ו done לאחר סיום הפעולה.

התמונות הן grayscale 8-bit, כלומר הערכים נעים בין 0-255, לכן data\_width=8.

בנוסף יש שימוש בפונקציית עצרת ליצירת הפילטר בעזרת משולש פסקל.

הקוד מכיל מכונת מצבים מסוג MOORE בה יש את המצבים:

**IDLE**- מצב התחלתי. יוצאים ממצב זה כאשר מכניסים בכניסה start.

**CALC\_KERNEL**- בו מייצרים את הפילטר SigmaMatrix. יצירת הפילטר קורת ע"י שימוש במשולש פסקל כמו שמתואר במבוא התיאורטי. בפעולת הנרמול יש שימוש בהכפלה ב 256 לפני החלוקה ובהמשך הקוד יש חלוקה ע"י הזזה 8 סיביות ימינה. דבר זה טוב לעבודה עם חומרה בכך שהוא מאפשר לנו להמשיך לעבור עם integer שיהיו fixed point ולא real בעזרת floating point.

**LOAD\_IMAGE**- בו לוקחים את הנתונים של התמונה בכניסה וטוענים אל memory.

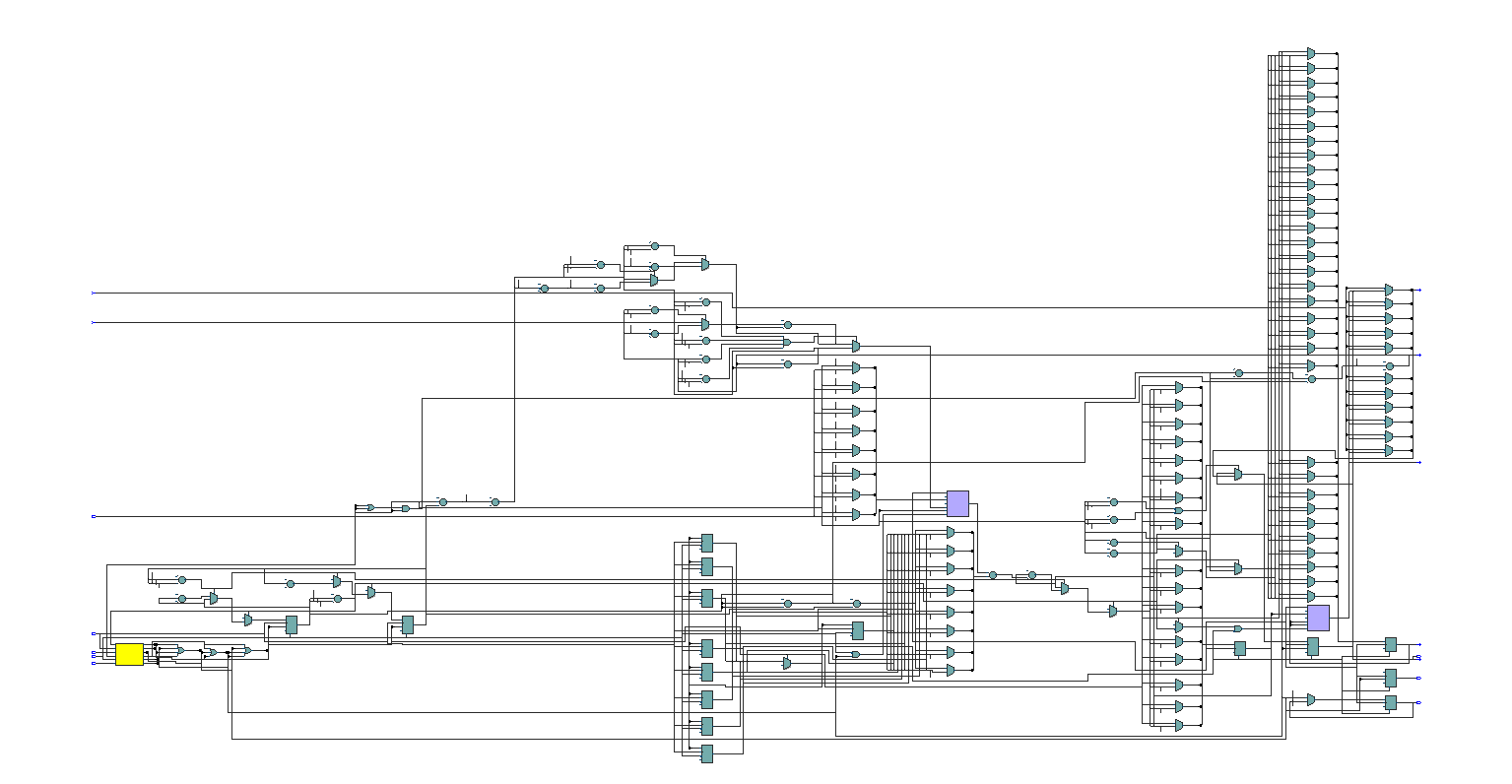
**CONVOLVE**- בו מבצעים את פעולת הקונבולוציה בין ערכי התמונה לפילטר. בפעולה זו ניתן תנאי שאנו לא נמצאים בקצוות. לאחר מכן מופיע החלק של הטיפול בקצוות. במידה ועברנו את השורות\עמודות מלמטה, האינדקס של השורה\עמודה מקבל את מינוס בהתחלה (מה שמחזיר אותו לחיובי. במידה ועברנו מלמעלה אנו מתחילים לספור מהמספר המקסימלי לאחור, כך אנו למעשה מבצעים את פעולת ה mirroring.

**OUTPUT\_RESULT**- בו תוצאת הקונבולוציה result מעודכן אל תוך המוצא image\_out.

**DONE**- שמצביע על כך שהפעולה הסתיימה ומחזיר אותנו למצב IDLE.

**פרויקט סיום - סכמת המערכת:**

**RTL**:



דיאגרמת מכונת מצבים:

תמונה שמכילה תרשים, עיגול, קו

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

תמונה שמכילה טקסט, צילום מסך, מספר, תצוגה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**הסבר על סכמת המערכת:**

**פרויקט סיום - קוד הבדיקה ה – TB(Test Bench):**

//-----------------------------------------------------

// TB

//-----------------------------------------------------

`timescale 100ns / 1ns

module GaussFilter\_tb;

parameter rows = 4;

parameter cols = 4;

parameter ksize = 3;

parameter data\_width = 8;

integer i,ii;

wire image\_out\_valid;

wire [data\_width-1:0] req;

wire [data\_width\*rows\*cols-1:0] image\_out;

wire done;

reg [7:0] address;

reg reset;

reg clk=0;

reg [data\_width\*rows\*cols-1:0] image\_in;

reg [data\_width-1:0] mem [0:(rows \* cols)-1] ;

reg image\_ready;

reg start;

assign req = mem[address];

initial begin

$readmemb("C:/files/image\_in.txt", mem); // memory\_list is memory file

end

always

#5 clk = ~clk;

GaussFilter

#(

.rows(rows),

.cols(cols),

.ksize(ksize),

.data\_width(data\_width)

)

GaussFilter\_inst

(

.clk (clk),

.rst (reset),

.start (start),

.image\_in (image\_in),

.image\_ready(image\_ready),

.image\_out (image\_out),

.done (done),

.image\_out\_valid (image\_out\_valid)

);

integer output\_file;

initial begin

output\_file = $fopen("C:/files/image\_out.txt", "w");

end

initial begin

address = 0;

reset = 1;

image\_ready = 0;

image\_in = 0;

#100

reset = 0;

#100

@(posedge clk)

for (i = 0; i < rows \* cols ; i = i +1 )begin

@(posedge clk)

address = address + 1;

image\_in[i\*8 +: 8] = req;//(7:0),(15:8)

end

image\_ready <= 1'b1;

start <= 1'b1;

@(posedge clk)

start <= 1'b0;

@(posedge clk)

@(posedge clk)

wait(done == 1'b1)

@(posedge clk)

$fwrite(output\_file, "%h\n", image\_out);

$fclose(output\_file);

end

endmodule

**הסבר על קוד הבדיקה ה – TB:**

קוד הבדיקה מורכב משעון בתדר של 1Mhz, reset אסינכרוני שני קבצים ו-instance למודל הראשי.

קובץ אחד מכיל שורה אחר שורה בצורה בינארית את הפיקסלים של התמונה המקורית:

תמונה שמכילה טקסט, צילום מסך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

הקובץ השני מכיל את הפיקסלים בצורה מחוברת והפוכה בייצוג הקסדצימלי:

תמונה שמכילה טקסט, צילום מסך, גופן, לבן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**פרויקט סיום - תוצאות סימולציה – דיאגרמת הזמנים:**

תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

קריאת הפיקסלים של התמונה לפני הפילטור

יצירת קובץ היציאה

**פרויקט סיום - סיכום ומסכנות מהעבודה של הפרויקט:**

תהליך בדיקת הפרויקט כלל ייצור תמונה כלשהי ויצירת קובץ הפיקסלים שלה בעזרת הפייתון.

לאחר הרצת הפרויקט שלנו למשך זמן מסוים בסימולציה בתוכנת המודלסים נלקח קובץ היציאה שהוכן על ידי סביבת הבדיקה לתוך קוד נוסף של פייתון שהפך את סדר הפיקסלים שהתקבלו.

השלב האחרון היה לקחת את הפיקסלים החדשים ודרך קוד פייתון לייצר תמונה חדשה ולראות עד כמה היא דומה לתמונה המפולטרת בעזרת פייתון וכמה היא מטושטשת(כמתבקש בפילטור גאוסי).

תמונה שמכילה טקסט, צילום מסך, סמל, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

הצלב השחור העליון הוא התמונה המקורית שממנה נלקחים הפיקסלים לקובץ הבדיקה בפרויקט שלנו.

התמונה שמתחת לצלב השחור היא התמונה לאחר הטשטוש הגאוסי לאחר פילטור בפייתון.

התמונה הבאה היא התמונה לאחר הטשטוש הגאוסי בוורילוג עם הפרויקט שלנו:

תמונה שמכילה צילום מסך, עיצוב, ריבוע

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

ניתן לראות שקיבלנו תוצאות דומות מאוד בהשוואה לפילטור גאוסי בפייתון. ההבדל נובע בעיקר מכך שבוורילוג יצרנו את הפילטר ע"י הערכים של משולש פסקל ולא על ידי חישוב של פונקציית גאוס כמו שנעשה בפייתון.

את המחברת פייתון ניתן לראות בקובץ נפרד.