УДК 004.92

Качурівський В.О.
Відокремлений підрозділ
Національного університету біоресурсів і природокористування України
«Бережанський агротехнічний інститут»

ПОБУДОВА АДАПТИВНИХ ТА ДИНАМІЧНИХ ДІАГРАМ ЗАСОБАМИ CANVAS API

У статті розглянуто JavaScript сценарій побудови діаграми засобами CANVAS API. Описано етапи створення стовпчикової діаграми у HTML-документі. Вирішено питання адаптування полотна побудови діаграми до ширини екрану. Визначено способи передачі числових даних для побудови графічних зображень, які автоматизують процес побудови інформаційного графіка.

***Ключові слова**: комп'ютерна графіка, JavaScript сценарій, побудова діаграм, інформаційна графіка.*

Постановка проблеми. Усе частіше в аналітичних матеріалах сайтів використовують супровід числових даних їх візуальним поданням (інфографіка) за допомогою різного роду графічних об'єктів, призначених для швидкого та доступного сприйняття усієї інформації комплексно. Для створення інформаційної графіки використовуються графічні редактори, спеціалізовані додатки та модулі прикладних програм, результатом роботи яких є збережені на зовнішніх носіях файли відповідного типу. Створені графічні зображення типу jpg, png, gif вкладаються у матеріали web-сторінки за допомогою тегу `<img>` для подальшого відображення на сайті.

Одним із способів візуалізації інформації, який набув широкого застосування, є використання різного виду діаграм: стовпчикова гістограма, кругова діаграма, двовимірний графік тощо. Популярним засобом створення діаграм є додаток Microsoft Office Excel. Створену та форматовану діаграму зберігають у файлі графічного типу для подальшого використання.

Поряд із простотою та перевагами цього способу він має ряд недоліків, а саме: створені діаграми є статичними, тобто зі зміною числової інформації графік незмінний; розмір файлу зображення є достатньо великим, тому з використанням декількох зображень значно збільшується час на завантаження сторінки із web-сервера; відкритим залишається питання адаптації відображення інфографіки до різних видів мобільних пристроїв та інтернет-браузерів.

Вирішити дану проблему можна із використанням графічних можливостей HTML5, зокрема програмованої графіки Canvas API.

Аналіз останніх досліджень і публікацій. У публікаціях Danny Goodman [3] та Володимира Дронова [5] розглянуто питання створення комп'ютерної графіки у HTML-документах за допомогою CANVAS. У працях Steve Fulton та Jeff Fulton [2] і Джоша Мариначи [4] детально висвітлено оператори для побудови геометричних фігур, які можна використовувати для комплексної побудови інфографіки. Основи роботи з елементом HTML5 `<canvas>` описано у публікаціях [1; 6]. На даний час практично відсутні публікації, в яких описані методи та способи адаптації полотна побудови графіка під розміри екрану, а також не розглянуті способи передачі до Javascript сценарію числових даних, на основі яких будується інформаційний графік.

Постановка завдання. Мета – визначити способи адаптації області побудови діаграм, створених за допомогою програмованої графіки Canvas API, до ширини екрану пристрою, на якому вона відображається. Розглянути шляхи передачі числових даних, на основі яких будується діаграма, до JavaScript сценарію для створення динамічної та актуальної інфографіки.

Виклад основного матеріалу дослідження. Розробниками HTML5 додано новий елемент `<canvas>`, який призначений для створення графічного полотна та побудови зображень за допомогою сценаріїв у JavaScript.

Основними етапами створення графіки є:

1. Створення графічного полотна у HTML-документі за допомогою парного тегу.

```
<canvas id="Your name"></canvas>
```

2. Написання JavaScript сценарію для побудови графічних зображень. Кожен сценарій, який забезпечує побудову певного виду діаграми, повинен мати свій унікальний ідентифікатор. Наприклад: `gistogram()`.

3. Виконання JavaScript сценарію побудови зображень під час активації документа.

```
<script>
//виклик сценарію побудови діаграми
window.onload = gistogram();
</script>
```

Розглянемо ці етапи побудови детальніше, на прикладі побудови вертикальної стовпчикової діаграми, на основі одного числового ряду.

Етап перший: створення полотна для графіки та адаптація його ширини.

Canvas використовує полотно, яке складається з множини графічних точок – пікселів. Розміри полотна в пікселях визначаються атрибутами `width` та `height` тегу `<canvas>`. За замовчуванням встановлюється ширина полотна – 300 px, а висота – 150 px. Задавши абсолютні розміри, ми прив'язуємо полотно до визначеного відображення на екрані. Під час перегляду сторінки на екранах мобільних пристроїв із меншою кількістю пікселів, ніж ширина полотна, зображення виходить за межі екрану. Постає питання, яким чином адаптувати ширину полотна до повноцінного відображення.

Одним із способів адаптації ширини полотна є зміна атрибута `width` тегу `canvas` до ширини контейнера, в якому відображається полотно програмним способом під час завантаження web-сторінки. Розробники сайтів застосовують різноманітні способи адаптації сайту до відображення на екранах різних пристроїв. Вказати універсальний ідентифікатор чи клас контейнера, за яким можна визначити реальну ширину у пікселях, яка відведена на відображення інформації, неможливо. Ми пропонуємо помістити тег `<canvas>` у контейнер `<div>` з унікальним ідентифікатором. Для прикладу: `id="canvas_for_drawing"`, який і слугуватиме відправною точкою для визначення ширини та адаптації полотна під його ширину програмним способом. Фрагмент HTML-документа буде таким:

```
<div id="canvas_for_drawing">
<canvas id="Your_name"></canvas>
</div>
```

Під час виклику JavaScript сценарію передбачено можливість передачі вхідних параметрів до самого коду. Таким способом передамо ідентифікатори контейнера та графічного полотна в код `gistogram ('canvas_for_drawing', 'Your_name')`. В ім'я сценарію побудови внесемо формальні змінні `div_id` та `canvas_id`, які будуть приймати відповідні значення. Активізація JavaScript сценарію побудови задається в такий спосіб:

```
window.onload = gistogram('canvas_
for_drawing',' Your_name');
```

До сценарію побудови записуємо відповідний фрагмент:

```
//ідентифікація полотна
var canvas = document.getElementById
(canvas_id);
var ctx = canvas.getContext ("2d");
```

Визначення ширини контейнера за формалізованою змінною `div_id` та встановлення нових параметрів проводимо так:

```
var cont= document.getElementById
(div_id);
// визначення ширини контейнера та
зменшення його ширини на 5%
var      W=cont.offsetWidth-cont.
offsetWidth*0.05;
var H=W/2;
// встановлення нових параметрів
полотна
canvas.width = W; canvas.height = H;
```

Отже, в одному документі можна застосувати декілька полотен та один JavaScript сценарій для побудови одного типу діаграм із різними числовими даними.

Етап другий: JavaScript сценарій для відображення графіки.

Цей етап потрібно розділити на декілька частин:

Частина 1: Побудова сітки діаграми (горизонтальних та/або вертикальних допоміжних ліній).

Частина 2: Задання кольору та числових даних для побудови діаграми.

Частина 3: Масштабування значень даних для побудови.

Частина 4: Побудова на полотні геометричних фігур.

Частина 5: Підписи осі та ряду даних.

Під час адаптації полотна побудови до розмірів контейнера було використано локальні змінні W та H – фактичні розміри canvas. Ці змінні будуть використовуватися під час побудови сітки та масштабування побудови геометричних фігур.

Для побудови сітки використано такий код:

```
// вертикальні лінії
ctx.beginPath();ctx.lineWidth=1;
for(var t=W/10; t<W; t+=W/10)
{ctx.moveTo(t,0);ctx.lineTo
(t,H-20);ctx.stroke();}
// горизонтальні лінії
ctx.beginPath();
for(var v=H/4; v<H; v+=(H-25)/4)
{ctx.moveTo(0,v);ctx.
lineTo(W,v);ctx.stroke();}
```

Десять допоміжних вертикальних та чотири горизонтальні лінії по всій ширині та усій висоті полотна. Змінивши значення знаменника у виразі `W/10` або `H/4`, змінимо кількість допоміжних ліній. Для підписів осі X зарезервовано 20 точок, тому H – 20.

Для реалізації *Частини 2* побудови діаграми необхідно передати до JavaScript сценарію код кольору та числові дані, на основі яких будуються прямокутники. Код кольору елементів діаграми передаємо безпосередньо під час виклику сценарію як один із вхідних параметрів.

Передачу числових даних можна реалізувати трьома способами:

1) передати конкретні числові дані безпосередньо під час виклику сценарію;

2) провести зчитування значень із файлу даних певного формату;

3) отримати дані з таблиці, яка розміщена в HTML-документі, інше.

Передачу кольору та числових даних безпосередньо в сценарій побудови діаграми під час його виклику проводять таким чином:

```
gistogram('canvas_for_drawing','
Your_name','#770000',20,30,50,40);
```

У цьому випадку передано колір #770000 та чотири числові значення 20, 30, 50 та 40.

Сценарій зберігає передані дані в асоційованому масиві `arguments`. Кількість переданих даних визначається конструкцією `arguments.length`. Індексація елементів масиву починається з нуля. Кількість елементів масиву 7. Передані числові дані в даному випадку знаходяться в елементах масиву `arguments[3], arguments[4], arguments[5], arguments[6].`

Передані числові дані сформуємо у масив y=[].

```
//формування числового масиву
var d=arguments.length;var y=[];
for(i=3;i<d;i++)y[i-3]=arguments[i];
```

Кількість числових даних `var n=d-3;`

Частина 3: Масштабування значень даних.

Для побудови прямокутників припустимо, що одиниця числа відповідає одному пікселю. Оскільки значення числових даних можуть бути більшими від фізичної висоти полотна у пікселях H, є необхідність ввести коефіцієнт масштабування k. Обчислення цього коефіцієнта проведемо таким чином: висоту полотна H поділимо на найбільше передане числове значення. Для визначення максимального скористаємося загальновідомим алгоритмом пошуку найбільшого елемента масиву.

```
max=y[0];
for(i=1;i<n;i++){if(y[i]>max)
max=y[i];}
k=(H-20)/(max+10);//коефіцієнт
масштабування
```

Частина 4: Графічна побудова геометричних фігур (елементів діаграми)

Ширину прямокутників визначаємо як `ww=W/n/2`, а крок між прямокутниками як `W/n.`

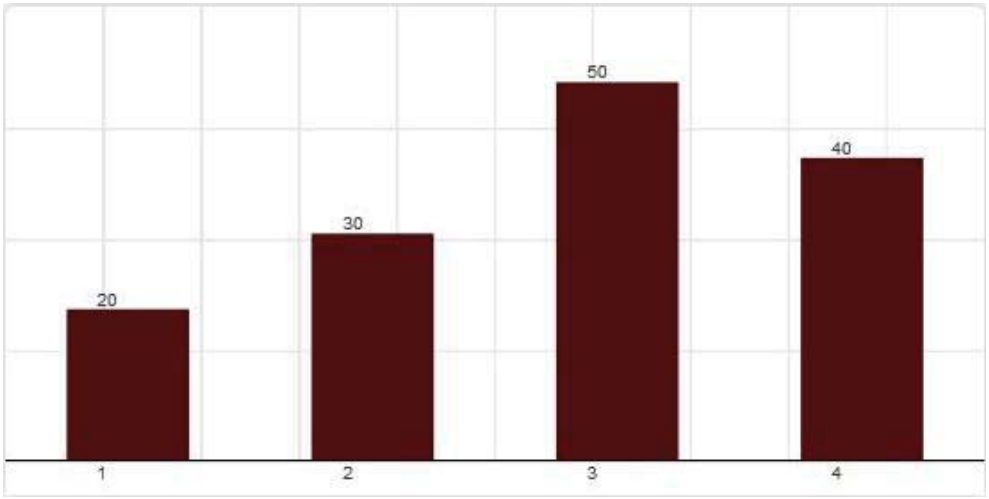

Рис. 1. Адаптована стовпчикова діаграма

Побудову прямокутників проведемо за допомогою циклу, використовуючи масив числових значень у.

```
var x=W/n/4;var ww=W/n/2;
var colorf= arguments[2];
for(i=0;i<n;i++)
{ctx.fillStyle = colorf;
ctx.fillRect    (x,H-y[i]*k-20,ww,
y[i]*k);
x+=W/n;}
```

Частина 5: Підписи осі та ряду даних.

Вісь OX будуємо наступним фрагментом

```
ctx.strokeStyle="rgb(1,1,1)";ctx.
beginPath();
ctx.moveTo(0,H-20);ctx.
lineTo(W,H-20);ctx.stroke();
```

Підписи осі X та ряду даних проведемо у цьому ж циклі за допомогою такої конструкції

```
ctx.fillStyle ='#333333';
ctx.fillText(y[i],x+ww/n,
H-y[i]*k-22);
ctx.fillText(i+1,x+ww/n,H-10);
```

У результаті ми отримаємо таке зображення (рис. 1.)

Результуючий JavaScript сценарій із використання зовнішнього файлу даних буде наступним.

```
/* Вертикальна діаграма*/
function      gistogram_d(div_id,
canvas_id,colorf,path){
//ідентифікація полотна за id
var canvas = document.getElementById
(canvas_id);
var ctx = canvas.getContext ("2d");
//вибір контейнера за id
var cont= document.getElementById
(div_id);
var      W=cont.offsetWidth-cont.
offsetWidth*0.05;
var H=W/2;
//встановлення параметрів canvas
canvas.width = W; canvas.height = H;
ctx.strokeStyle="rgb(231,231,231)";
//сітка
ctx.beginPath();ctx.lineWidth=1;
for(var t=W/10; t<W;t+=W/10)
{ctx.moveTo(t,0);ctx.lineTo
(t,H-20);ctx.stroke();}
ctx.beginPath();
for(var v=H/4; v<H; v+=(H-25)/4)
{ctx.moveTo(0,v);ctx.lineTo
(W,v);ctx.stroke();}
//формування числового масиву
var i,max,k,d;var y=[];
var d=arguments.length;var y=[];
for(i=3;i<d;i++) y[i-3]=arguments[i];
//закінчення формування числового
масиву
var n=d-3;max=y[0];
for(i=1;i<n;i++){if(y[i]>max)
max=y[i];}
k=(H-20)/(max+10);
var x=(W/n)/4;var ww=W/n/2;
for(i=0;i<d;i++){
ctx.fillStyle = colorf;
ctx.fillRect (x,H-y[i]*k-20,ww,y[i]*k);
ctx.fillStyle ='#333333';
ctx.fillText(y[i],x+ww/n,
H-y[i]*k-22);
ctx.fillText(i+1,x+ww/n,H-10);
x+=W/n;}
ctx.strokeStyle = "rgb(1,1,1)";ctx.
beginPath();
ctx.moveTo(0,H-20);ctx.lineTo(
W,H-20);ctx.stroke();}
```

Кожному JavaScript сценарій для побудови певного виду діаграми необхідно присвоювати унікальний ідентифікатор, що дозволить організувати їх у зовнішній файл та підключати за необхідністю до конкретної web-сторінки для застосування.

Висновки. Розглянуті питання адаптації полотна графіки до ширини екрану, способи передачі даних до JavaScript сценарій для побудови актуальних та динамічних діаграм, способи масштабування числових даних не вичерпують усіх можливих підходів до даної проблеми. Оскільки компонентами діаграми є інші інформаційні елементи, то JavaScript сценарій побудови має достатнє широке поле для вдосконалення та розвитку. Нами розроблено сценарії імплементації у HTML-документ кругової, секторної, горизонтальної діаграми та площинного графіка. Потребують подальшої розробки інші способи графічного відображення даних, створення нестандартних діаграм, вивчення питання зчитування числових даних з інших типів файлів. Також заслуговують уваги питання написання коду JavaScript сценарію для анімації елементів діаграми.

Список літератури:

1. HTML5 Canvas. URL: https://www.tutorialrepublic.com/html-tutorial/html5-canvas.php (дата звернення: 15.01.2018).
2. Steve Fulton, Jeff Fulton. HTML5 Canvas. O'Reilly Media, 2011. 654 p.
3. Д. Гудман, JavaScript и DHTML. Сборник рецептов. Для профессионалов. Москва, 2015. 523 с.
4. Джош Мариначи. Основы рисования. URL: https://webref.ru/dev/canvasdeepdive/chapter01 (дата звернення: 15.01.2018).
5. Дронов В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. Москва, 2013. 416 с.
6. Элемент Canvas HTML5. URL: https://msdn.microsoft.com/uk-ua/gg589510(v=vs.85) (дата звернення: 21.01.2018).

ПОСТРОЕНИЕ АДАПТИВНЫХ И ДИНАМИЧЕСКИХ ДИАГРАММ СРЕДСТВАМИ CANVAS API

В статье рассмотрен JavaScript сценарий построения диаграммы при помощи CANVAS API. Описаны этапы создания столбиковой диаграммы в HTML-документе. Решен вопрос адаптации полотна построения диаграммы к ширине экрана. Определены способы передачи числовых данных для построения графических изображений, которые автоматизируют процесс построения информационного графика.

Ключевые слова: *компьютерная графика, JavaScript сценарий, построение диаграмм, информационная графика.*

CONSTRUCTION OF ADAPTIVE AND DYNAMIC DIAGRAMS BY MEANS OF CANVAS API

The scientific article is considered the JavaScript screenplay for constructing a chart with CANVAS API. Describes the stages of creating a chart in the HTML-document. The question of adapting the canvas for constructing a diagram to the width of the screen is solved. Determined methods of numerical data transmission for constructing graphic images, which automate the process of constructing an information graph.

Key words: *computer graphics, Javascript screenplay, diagram design, information graphic.*